Aalto University
School of Business

SEMI-HIGH FREQUENCY PORTFOLIO OPTIMIZATION WITH DEEP REINFORCEMENT LEARNING

Analysis of cryptocurrency markets from 2016 to 2019

Master's Thesis

Nikolai Linnansalo
Aalto University School of Business
Department of Finance

Spring 2019

Aalto University
School of Business

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi

Abstract of master's thesis

Author Nikolai Linnansalo

Title of thesis Semi-High Frequency Portfolio Optimization with Deep Reinforcement Learning – Analysis of Cryptocurrency Markets From 2016 To 2019

Degree Master of Science in Economics and Business Administration

Degree programme Finance

Thesis advisor(s) Sami Torstila

Year of approval 2019 **Number of pages** 98 **Language** English

Abstract

This thesis explores machine learning techniques in algorithmic trading. We implement a trading computer program that balances a portfolio of cryptocurrencies. We try to outperform an equally weighted strategy. As our machine learning technique, we use deep reinforcement learning.

Cryptocurrencies are digital mediums of exchange that use cryptography to secure transactions. The most well-known example is Bitcoin. They are interesting to analyze due to high volatility and lack of previous research. The availability of data is also exceptional.

We introduce an algorithmic trading agent – a computer program powered by machine learning. The agent follows some pre-determined instructions and executes market orders. Traditionally a human trader determines these instructions by using some technical indicators. We instead give the trading agent raw price data as input and let it figure out its instructions. The agent uses machine learning to figure out the trading rules.

We evaluate the performance of the agent in seven different backtest stories. Each backtest story reflects some unique and remarkable period in cryptocurrency history. One backtest period was from December 2017 when Bitcoin reached its all-time high price. Another one is from April 2017 when Bitcoin almost lost its place as the most valued cryptocurrency. The stories show the market conditions where the agent excels and reveals its risks.

The algorithmic trading agent has two goals. First, it chooses initial weights, and then it rebalances these weights periodically. Choosing proper initial weights is crucial since transaction costs make trade action costly. We evaluate the trading agent's performance in these two tasks by using two agents: a static and a dynamic agent. The static agent only does the weight initialization and does not rebalance. The dynamic agent also rebalances. We find that the agent does a poor job in choosing initial weights.

We also want to find out the optimal time-period for rebalancing for the dynamic agent. Therefore, we compare rebalancing periods from 15 minutes to 1 day. To make our results robust, we ran over a thousand simulations. We found that 15 – 30 minutes rebalancing periods tend to work the best.

We find that the algorithmic trading agent closely follows an equally weighted strategy. This finding suggests that the agent is unavailable to decipher meaningful signals from the noisy price data. The machine learning approach does not provide an advantage over equally weighted strategy. Nevertheless, the trading agent excels in volatile and mean reverting market conditions. On these periods, the dynamic agent has lower volatility and a higher Sharpe ratio. However, it has a dangerous tendency of following the looser.

Our results contribute to the field of algorithmic finance. We show that frequent rebalancing is a useful tool in the risk management of highly volatile asset classes. Further investigation is required to extend these findings beyond cryptocurrencies.

Keywords machine learning, deep learning, reinforcement learning, algorithmic trading, algorithmic finance, cryptocurrencies, Bitcoin, Ethereum, Ripple

Aalto-yliopisto
Kauppakorkeakoulu

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi

Maisterintutkinnon tutkielman tiivistelmä

Tekijä Nikolai Linnansalo

Työn nimi Lyhyen intervallin salkunhallinta syvällä vahvistusoppimisella – Katsaus kryptovaluuttamarkkinoilta 2016-2019

Tutkinto Kauppatieteiden maisteri

Koulutusohjelma Rahoitus

Työn ohjaaja(t) Sami Torstila

Hyväksymisvuosi 2019 **Sivumäärä** 98 **Kieli** Englanti

Tiivistelmä

Tämä tutkielma käsittelee koneoppimista algoritmisessa kaupankäynnissä. Toteutamme kryptovaluuttasalkkua hallitsevan tietokoneohjelman. Haluamme selvittää sen, että onko koneoppiva salkku tuottavampi kuin tasapuolisesti painotettu salkku. Menetelmämme on syvä vahvistusoppiminen.

Kryptovaluutat ovat digitaalisia vaihdannan välineitä, jotka käyttävät kryptografiaa kauppojen todentamiseen. Tunnettu esimerkki on Bitcoin. Ne ovat kiinnostavia tutkia algoritmisen kaupankäynnin näkökulmasta korkean volatiliteetin, tutkimuksen puutteen sekä aineiston saatavuuden vuoksi.

Esittelemme algoritmisen kaupankäyntiagentin, joka on pohjimmiltaan koneoppimisella toimiva tietokoneohjelma. Agentti seuraa ennalta määrättyjä ohjeita ja suorittaa simuloituja kauppoja. Perinteisesti ihmiskauppiaat ovat määrittäneet nämä ohjeet käyttäen hyväksi teknisiä indikaattoreja. Agenttimme sen sijaan oppii itsenäisesti ohjeensa käyttäen tietonaan vain raakaa hintadataa.

Arvioimme agenttimme suorituskykyä seitsemällä eri ajanjaksolla. Jokainen ajanjakso heijastaa ainutlaatuista ja merkittävää ajankohtaa kryptovaluuttojen historiassa. Yksi ajanjakso on joulukuussa 2017, jolloin Bitcoin saavutti kaikkien aikojen korkeimman arvonsa. Toinen on huhtikuussa 2017, jolloin Bitcoin lähes menetti paikkansa arvostetuimpana kryptovaluuttana. Ajankohtien erilaisuus mahdollistaa meitä näyttämään missä olosuhteissa agentti on käyttökelpoinen.

Agentillamme on kaksi tavoitetta. Ensiksi se valitsee alkupainot ajanjaksolle ja toisekseen se tasapainottaa salkkua tasaisin väliajoin. Sopivien alkupainojen valinta on tärkeää, koska siirtokustannukset tekevät kaupankäynnistä kallista. Arvioimme agenttimme suorituskykyä näissä tavoitteissa jakamalla sen kahteen eri osaan: staattiseen ja dynaamiseen agenttiin. Staattinen agentti valitsee vain alkupainot eikä tasapainota salkkua lainkaan. Dynaaminen agentti myös tasapainottaa salkkua. Tulostemme mukaan agentti on huono valitsemaan alkupainoja.

Haluamme myös selvittää mikä on sopiva väliaika salkun tasapainottamiseen. Tämän vuoksi vertaamme tasapainotusaikoja 15 minuutista yhteen päivään. Ajamme satoja simulaatioita jokaiselle ajanjaksolle, jotta tuloksemme olisivat vakaita. Huomaamme, että 15 - 30 minuutin tasapainotusjaksot toimivat parhaiten.

Havaitsemme, että agenttimme seuraa tarkasti tasapainoisesti painotettua strategia. Tämä viittaa siihen, että koneoppimismalli ei kykene löytämään merkityksellisiä signaaleja hintadatasta. Agentti kuitenkin pärjää hyvin, kun markkinoiden hintaheittelyllä on taipumuksena palata keskiarvoon. Dynaamisella agentilla on huomattavasti pienempi volatiliteetti ja korkeampi Sharpe-tunnusluku kun markkinat ovat erityisen epävakaita. Agentilla on vaarallinen taipumus seurata häviäjiä.

Tuloksemme lisäävät tietämystä algoritmisessa rahoituksen alalla. Näytämme, että usein tapahtuva tasapainotus on hyödyllistä korkean volatiliteetin riskienhallinassa. Emme kuitenkaan väitä, että nämä tulokset ovat suoraan sovellettavissa muihin omaisuuslajeihin.

Avainsanat koneoppiminen, syväoppiminen, vahvistusoppiminen, algoritminen kaupankäynti, algoritminen rahoitus, kryptovaluutat, Bitcoin, Ethereum, Ripple

Table of Contents

List of figures

List of tables

1. Introduction

1.1. Motivation

This master's thesis demonstrates a continuous portfolio optimization strategy. The strategy uses machine learning models that empower an online trading agent. The trading agent continuously learns and adjusts itself based on new data observations. More specifically, the agent is based on reinforcement learning. It makes trading actions based on a deep learning policy. These stochastic search techniques have proven to be promising in prior research. We borrow the term agent from reinforcement learning literature. It roughly refers to a computer program that utilizes reinforcement learning.

The algorithmic trading agent is a computer program that periodically performs transactions. Already for decades, the automating of the order flow in financial markets has proven to be worthwhile. However, most of these trading programs have relied on hand-crafted rules. The writer of the program has based the rules on some technical indicators. In this thesis, we are going to let the trading agent discover the trading rules by itself with ML methods. The powerful ML methods used in this study have been proven to excel in tasks that have been considered very hard for computers.

During the recent decade, Deep learning models (DL) have experienced a renaissance. Nowadays, DL models are used in various intelligent applications such as Google Search and Translate. The key idea of DL is to train a vast artificial neural network to find meaningful patterns from noisy data. Research on similar architectures goes back to the 1940s (McCulloch and Pitts, 1943). However, only recent technological advancements have made these approaches practically feasible. A significant recent merit of DL is achieving the average bilingual human level in machine translation (Wu et al., 2018). DL models are also superhuman in cancer detection (Bychkov et al., 2017).

Reinforcement learning (RL) has been researched from at least the 1960s (Waltz et al., 1965). The fundamental idea of reinforcement learning is to optimize an agent to find optimal actions in some noisy environment. The agent interacts with its environment and learns by trial and error. Recently RL models have gained widespread attention due to the superhuman performance in the game of Go (Silver et al., 2016). Most modern RL implementations utilize DL to determine which action to take. We will call these models

deep reinforcement learning (DRL) models. Nevertheless, DL and RL are distinct technologies. Both systems can be implemented independently from the other.

There are some relevant portfolio management research papers that utilize these technologies. Heaton, Polson, and Witte used DL models to try to beat IBB, a Nasdaq Biotechnology ETF, with promising results (Heaton et al., 2016). There are at least two studies that utilize a hybrid model where both RL and DL used. In a 2017 study, Jiang et al. optimized a cryptocurrency portfolio with suspiciously spectacular results. (Jiang et al., 2017). We will follow the framework of Jiang et al. in this study. Also, Deng et al. explored these models with S&P data (Deng et al., 2017).

Now is a highly appealing time to research cryptocurrencies from a portfolio management point of view. Firstly, this type of analysis has only become feasible in recent years. During the past few years, credible competition for Bitcoin has emerged. Today many alternative cryptocurrencies (i.e.: altcoins) have multi-billion dollar market caps. Consequently, more assets can be used to compose a reasonable portfolio than before. Secondly, this is the first opportunity to study cryptocurrencies on a bear market. The last time the cryptocurrency market was bearish was from the beginning of 2014 to the summer of 2015. Back then, there were only a handful of relevant competitors to Bitcoin. Nevertheless, the ML models built here are general. They could be applied to any other asset class if relevant data is available.

We test the agent in seven different backtests stories. These stories demonstrate different market conditions. The stories reveal the conditions where our agent thrives and loses. Our results indicate that our agent converges to an equally weighted strategy. The agent balances itself periodically. We find out that the dynamic agent excels in mean reverting market conditions. It strictly outperforms its benchmarks in three of the backtests. In one backtest it greatly reduces volatility on the cost of return and in another one vice-versa. We compared the agents' performance in five rebalancing periods ranging from 15 minutes to 1-day. We find evidence that 30 minutes to 2 hour rebalancing periods are optimal for our agent. We describe these short periods to be of semi-high frequency. The term high-frequency trading refers to even shorter periods: micro- or milliseconds.

This study contributes to the field of algorithmic finance. More specifically, we contribute to the field of agent-based finance. To the best of our knowledge, this is the first study that explicitly demonstrates how a DRL agent balances its weights. Our study is the first one

with such a comprehensive backtest setting. Finally, this is the first study that directly compares the impact of different rebalancing periods.

1.2. Cryptocurrencies

"The one thing that's missing but that will soon be developed is a reliable e-cash, a method whereby on the internet you can transfer funds from A to B without A knowing B or B knowing A, the way in which I can take a twenty-dollar bill and hand it over to you and there is no record of where it came from and you may get that without knowing who I am."

Milton Friedman, 1999

1.2.1. What are cryptocurrencies

In 2008 Satoshi Nakamoto released the Bitcoin whitepaper that envisioned a new kind of medium of exchange: cryptocurrencies. He introduced the blockchain technology, a continuously growing list of records (called blocks) which are linked and secured using cryptography (Nakamoto, 2008). Bitcoin is the first and most well-known example of a protocol that utilizes the blockchain technology, but there are many others (for example Ethereum, Litecoin, and Ripple).

While there are crucial differences in the protocols, they share the same fundamental concepts. They all rely on a distributed timestamp service that tackles the double-spending problem by verifying each transaction in a decentralized network of nodes. Anyone with a computer and internet connection can make transactions, verify transactions and also mine the cryptocurrencies. In the rest of this section, we will use Bitcoin as an example.

Instead of relying on a trusted third party, Bitcoin utilizes cryptography to secure transactions by a process called mining. The miners' task is to encapsulate transactions to a block while finding a solution to a cryptographic puzzle. Briefly, the puzzle is to find a predetermined amount of leading zeros for an SHA-256 hash of a block. The appropriate amount of leading zeros is found by systematically trying out different cryptographic nonces, which are basically just random numbers. Once a miner finds a valid nonce, the newly found block is automatically added to the blockchain and declared to the Bitcoin network. As every participant in the network agree that the longest chain is king, all miners start building on top of the newly found block and try to search for a new block. Since each block

cryptographically refers to all its predecessors, it becomes exponentially harder to manipulate a block as time passes by (Nakamoto, 2008).

The miners are incentivized to donate computation time to the network with block rewards. A block reward consists of newly created Bitcoins that are automatically awarded by the Bitcoin software to the miner who successfully adds a new block to the blockchain. The Bitcoin software ensures that a new block is found on average every 10 minutes. The software adjusts the difficulty of finding a block based on how much computation power is donated to the network by miners. (Nakamoto, 2008)

Bitcoin is designed to be borderless internet money. The Bitcoin software completely controls its behavior including monetary policy. Consequently, Bitcoin has highly predictable inflation. The Bitcoin software adds new Bitcoins to the ecosystem only via block rewards, and the software halves the block rewards after every 210 000 blocks (approximately four years). The current block reward (April 2019) is 12.5 Bitcoins. However, there is a limitation on how small units a single Bitcoin can be split into (0.00000001 BTC). Therefore the last Bitcoins will be mined in the year 2140.

Due to the price increase of cryptocurrencies, mining has become very popular. The colossal amount of hash power (i.e., computing power donated to the network) donated to the network by miners makes compromising the network almost impossible. Nevertheless, it is theoretically possible to compromise the Bitcoin network by manipulating the new blocks added to the blockchain by an assault called a 51% attack. Manipulating a block would require controlling 51% of the hash power for at least 30 minutes.

However, Bitcoin mining's' estimated energy consumption is enormous (29.05 TWh/year, a value larger than the energy consumption of Slovakia and Serbia). Furthermore, the network would detect malicious activity and probably agree to cancel any transactions made during the attack. As this would be very costly to the attacker and deliver very little value, it makes more economic sense to mine Bitcoin than to try to compromise the network. However, a threat of a 51% attack is real in smaller cryptocurrencies which have a smaller hash power.

Cryptocurrencies have many benefits over traditional currencies such as fast, inexpensive and pseudonymous global transactions. Nevertheless, they still have minimal real-life use cases. While blockchain technology has the potential to do wonderful things,

cryptocurrencies' value is still primarily driven by speculation. It is still unclear what the competitive advantage of cryptocurrencies is from a consumer or institutional perspective.

While it is unclear that what role cryptocurrencies will take in the modern economy, it is evident that there will not be a single cryptocurrency that dominates the market. At the end of the day, cryptocurrencies are just software, and it is impossible for a software application to excel in multiple problem domains. For example, a cryptocurrency cannot be a highly-secure store of value and a platform for fast-payments at the same time. Therefore cryptocurrencies are already now highly specialized in fields such as privacy (Monero), fast payments (Stellar), running legal contracts in a blockchain (Ethereum) and business applications (Ripple). There are currently over 1500 cryptocurrencies traded in tens of exchanges.

1.2.2. Cryptocurrencies in portfolio optimization

Cryptocurrencies have some very attractive properties as research data. Firstly, there is a lot of good quality data available for free. Cryptocurrencies are traded 24 hours a day all year round. Most exchanges such as Poloniex and Binance provide free APIs (Application programming interfaces) that provide access to data on micro time intervals as 5 minutes. These APIs also support algorithmic trading.

Secondly, many cryptocurrencies are listed on multiple exchanges. This enables to research them as cross-listed instruments and remove exchange-specific variance. Furthermore, a predictive model can be trained on the data from one exchange and validated on another.

Thirdly, due to the novelty of the phenomena, cryptocurrencies have not yet been researched a lot from a portfolio management perspective. It can be argued that such research has not even been sensible before 2017 due to small trading volumes of most cryptocurrencies.

Finally, since the total market cap of all cryptocurrencies is still relatively small ($176 billion [April 2019]) and the prices are mostly driven by speculation, some interesting correlations might be found. However, it is unclear how well found behaviors can be scaled to other asset classes. Furthermore, the found patterns might quickly vanish as the cryptocurrency markets develop.

1.3. Machine learning in finance

The purpose of this section is to briefly introduce machine learning from a financial standpoint. This is to give the reader some context before introducing the actual model we are going to use. The model used by this study will be introduced in chapter 4.

First, we will explain what machine learning is and what kind of problems it can solve. Then we will go through how machine learning is used in hedge funds. Finally, we will discuss the limitations of machine learning methods.

1.3.1. What is machine learning

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

Tom M. Mitchell, pioneer of machine learning

Machine learning (ML) models are predictive statistical models that are trained with historic data. While ML models have very different learning strategies and goals, they all share this common characteristic of a training phase with historic data in order to make predictions of future data points.

The most simple machine learning model is the well-known linear regression model, where a linear relationship is modeled between a dependent variable y and one or more explanatory variables X. The parameter vector β is trained with data to make predictions of new observations:

$$y = X\beta + \varepsilon$$

, where ε is an unobserved noise term we wish to minimize. While linear regression meets the broad definition of machine learning, it is rarely considered as a machine learning algorithm due to its simplicity.

ML models are very adaptive, they are utilized in a wide range of contexts including finance. The most common machine learning tasks are regression, classification, and clustering tasks. They all have relevant financial use cases:

- **Regression:** Estimating a continuous dependent value based on one or several independent values.
    - Example: Predicting the exact price of an asset based on historical data
    - Example ML models: Linear regression, Random Forest Classifiers, Artificial Neural Networks
- **Classification:** Predicting a class where a new observation belongs to.
    - Example: Predicting whether an assets' price will go up or down during the next week
    - Example ML models: Logistic regression, Support Vector Machines, K-nearest neighbors
- **Clustering:** A task of grouping observations based on similarity. These models are often used in data mining to find unexpected structure in chaotic data. The task of the algorithm is to find clusters that explain most of the variance.
    - Example: Clustering equities to groups based on a chaotic dataset
    - Example ML models: K-means clustering, Self-organizing maps

This thesis explores machine learning techniques in portfolio optimization. This problem is *a regression task* since the goal is to find optimal asset weights which are continuous values. As will be demonstrated, it is not enough to get an accurate estimate of future prices. The estimations of the future prices need to be converted in to actual market actions. In our case the market action will consist of rebalancing to new portfolio weights. While rebalancing, we need to consider factors such as previous asset weights, minimum transaction lots and transaction fees among others. These methods are discussed in detail in chapter 4.

1.3.2. Machine learning in hedge funds

"If computing power and data generation keep growing at the current rate, then machine learning could be involved in 99 percent of investment management in 25 years. It will become ubiquitous in our lives. I don't think that machine learning is the answer to everything we do. It just can make us better at a lot of things that we do."

Luke Ellis, CEO of Man Group Plc

Machine learning techniques are utilized by quantitative analysts in various hedge funds such as Man Group, Two Sigma, and Goldman Sachs Group Inc. While the final trading

strategies are usually made by humans, ML models enable the traders to make more informed trading decisions.

ML models mainly outperform human analysts by scale. A single team of quantitative analysts can design an ML algorithm that analyzes enormous amounts of data which is practically impossible to be analyzed manually. For example, an ML model can forecast the earnings of hundreds of thousands of companies based on various data types such as textual data, images, and video.

Furthermore, ML models are capable of finding undiscovered patterns in the stock market by exploiting new data sources altogether. For example, social media data and news can be analyzed to decipher investor sentiment. Job posts can be analyzed to reveal upcoming strategic changes in companies. Satellite images can be analyzed to estimate production amounts of a given company.

1.3.3. Challenges of machine learning in portfolio optimization

While machine learning techniques have their merits, they also have serious limitations when applied to financial data. These limitations include the noisiness of the data, lack of capability to anticipate black swans, and the lack of transparency of the ML models. Furthermore, the ML hedge funds are yet to beat the S&P 500.

While most ML use cases deal with static data (such as categorizing whether a picture is a dog or a cat) financial data is noisy. In static use cases, the performance of an ML model improves when more data is added to the model. However, financial ML models will not necessarily get more accurate. Recent data matters more than older data.

An ML model will most likely perform terribly in radically changed market conditions. The process of training an ML model consists of giving the model historic training data in order to make predictions of new data points. However, each financial crisis has a unique factor related to it and therefore the model cannot be trained beforehand to anticipate it. Therefore it is improbable that ML models will be given the final say on trading decisions any time soon.

The most powerful machine learning models such as deep learning models are very hard to understand. Since the training process involves stochasticity, the quantitative analyst who has written the code cannot really tell why the models behave in a way it does. By the nature

of the model, it is very difficult to say why a model has come up with a certain recommendation. Even if they would work well most of the time, this opacity makes it hard to trust these kinds of systems. Their limitations are difficult to comprehend. Deep learning models are further discussed in chapter 4.

Ultimately ML models will be judged whether they make money or not. While the Eurekahedge AI Index, a broad measure for the performance of AI utilizing hedge funds, beats most hedge funds, it has lagged behind the S&P 500 from the year 2010. (Figure 1)

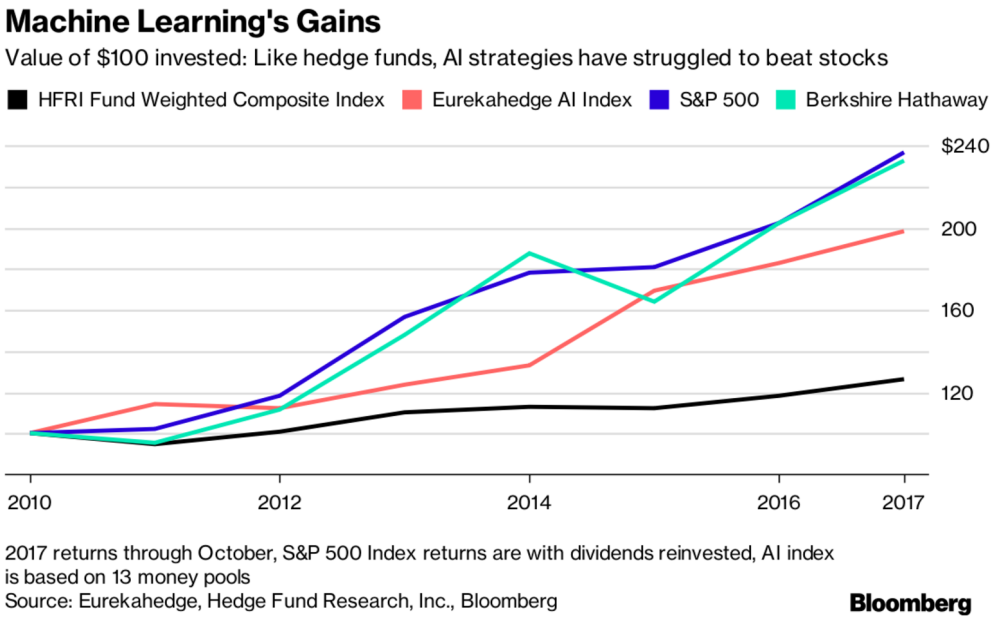


Figure 1: Machine Learning's Gains (from bloomberg.com)

1.4. Structure of the thesis

The structure of the thesis is the following. Firstly we are going to formalize the goals of this study by presenting our research questions. After that, we will go through the relevant prior literature which turns out to be scarce in availability. Then we will present the methodology we used: the deep reinforcement learning agent. Next, we explore and analyze the performance of the agent in various backtests. After that, we discuss what we learned from this study and suggest further research areas. The last chapter concludes.

2. Research questions

Prior research suggests that there is enough predictability in cryptocurrency markets that machine learning models can capture and exploit. We explore this further by managing a cryptocurrency portfolio with a deep reinforcement learning (DRL) agent. The agent's return consists of two components: the initial portfolio weights and the trading action. We want to make the impact of these components explicit. Therefore we implement two agents: a static and a dynamic agent. The static agent only initializes the portfolio weights at the beginning of the backtest period. The dynamic agent takes those initial weights and rebalances the portfolio periodically.

We evaluate the performance of the agents' on seven backtests periods that demonstrate different market conditions. We benchmark the performance against an equally weighted strategy. We seek to maximize risk-adjusted return.

Since neither the static agent nor the equally weighted strategy rebalances during the backtest period, their performance is directly comparable. Their difference purely stems from the DRL agent's capability to initialize good portfolio weights. Therefore we present our first research question:

> **[H1] Successful weight initialization:** The static agent beats an equally weighted strategy on risk-adjusted return.

The difference between the performance of the dynamic and static agent is purely based on the added value of trading action. Therefore we present our second research question:

> **[H2] Successful trading action:** The dynamic agent beats the static agent on risk-adjusted return.

Finally, we compare the performance of the dynamic agent on five different rebalancing period lengths: 15 minutes, 30 minutes, 2 hours, 4 hours and 1 day. We expect that there will be considerable value creation difference between these period lengths. We expect to find the trading period length that creates most value.

> **[H3] Rebalancing period effect:** We are able to identify a range of trading period lengths that creates most value.

3. Financial literature review

The scientific study of machine learning consists of numerous different types of predictive statistical models that aim to find meaningful patterns from a training set which generalize to an unseen test set. Many of these machine learning algorithms have also been researched from a portfolio optimization perspective. Most often the goal is to optimize expected risk-adjusted return. The approaches are commonly based on modern portfolio theory (Markowitz, 1952).

The purpose of this chapter is to summarize some of the previous attempts where machine learning was implemented in a portfolio optimization context. Firstly, we will discuss genetic algorithms (GAs). GAs are interesting, since they are similar to DL models at least in two aspects: 1) they both are inspired by biology and 2) they both excel with incomplete information and where the full set of solutions is too large to be completely sampled. They also resemble reinforcement learning algorithms since both machine learning approaches learn by trial and error. Reinforcement learning will be discussed in chapter 4.

Secondly, we will discuss the deep learning portfolio optimization literature and justify why these machine learning algorithms are worthy of further analysis.

Both GA and DL approaches rely on finding technical patterns from the price data. However, it is also interesting to consider whether it is possible to find fundamentals that drive cryptocurrency values. Therefore we will lastly discuss a paper by Bhambhwani et al. (2019). The paper finds evidence that cryptocurrency prices are driven by two fundamentals: computing power and adoption level.

3.1. Genetic algorithms in portfolio optimization

Biological evolution is a stochastic process where living organisms find optimized adaptation strategies in varying environments. This process can be mimicked with computers to find unforeseen solutions to arbitrary optimization problems, including portfolio optimization. Genetic algorithms were first introduced by Holland (1975) and have since become one of the most popular machine learning algorithms. They are capable of efficiently exploring the problem space and find near-optimal solutions. Moreover, they only need to compute a fraction of the whole problem space.

Similar to biological evolution, the GA process iterates through generations with a trial and error strategy. In each generation, the fitness of candidate solutions is evaluated and the most fit solutions are let to breed. The key part of the algorithm is the *fitness function*, a problem specific function that drives the optimization process.

In finance literature, genetic algorithms have mostly been researched from the perspective of Markowitz' famous Modern Portfolio Theory (Markowitz, 1952). In this setting the fitness function is set to minimize the risk given some fixed return. Consequently the process comes up with an ensemble of mean-variance optimized portfolios.

Lin and Liu showed that genetic algorithms could be used to compute portfolios with high mean-variance efficiency. With their model, a near-optimal portfolio could be computed while considering minimum transaction lots (Lin et al., 2007). A drawback in their model is that it is static: while their model composes mean-variance optimized portfolios, they are only momentarily optimized as they do not have any trading logic.

A more dynamic approach to the portfolio optimization problem was presented by Aranha and Iba. They introduced a hybrid model that combines a genetic algorithm with a local search process. The approach is capable of rebalancing portfolios while considering trading costs. Their dataset included Nasdaq and S&P 500 data from 2007-2008 and their active trading strategy was remarkably robust to the crashes during the volatile period. (Aranha et al., 2008)

Also purely GA driven active management strategies have been researched. Yan, Sewell and Clack proposed a genetic algorithm that uses a fitness function that maximizes the Sharpe ratio. Furthermore they used a clever voting scheme to make the GA more robust: three different GA's were trained during three different market environments: "bull", "bear" and "volatile". Their research was performed on equities in the Malaysian stock exchange. This exchange is quite volatile; a common feature in emerging markets. Their voting GA was able to beat the index during their rest period in 2001-2003. (Yan et al., 2008)

Another GA driven active management strategy was presented by Gorgulho, Neves and Horta. They proposed a stock picking investment simulator that utilizes technical indicators such as Exponential moving average (EMA), Relative Strength Index (RSI) and Moving Average Convergence Divergence oscillator (MACD). Their GA model maintains a population of portfolios and makes trades weekly based on the technical indicators. The

technical indicator signals were discretized to four categories: (very high, high, low and very low) and the thresholds were predetermined by expert knowledge. The job of the GA model was to *maximize return (fitness function)* by learning the respective predictive power of each technical indicator during the test period. Promisingly, their model was able to beat the Dow Jones Industrial Average Index on the test period from 2003-2009. (Gorgulho et al., 2011)

A recent study by Ha and Moon (2018) used GAs to mining useful signals based on technical indicators on cryptocurrency data. They added a separate trade simulator to actually implement these signals to a trading strategy. They claimed that their simulation yielded better results than a buy-and-hold strategy. However they did not provide details on how the technical indicators were actually evaluated. (Ha and Moon 2018)

3.2. Deep learning in portfolio optimization

Deep learning (DL) models are machine learning models that are loosely inspired by the architecture and functions of human brains. Similarly to genetic algorithms, they are capable of finding patterns in data without explicitly being told what to find while exploring just a small part of a huge search space. Due to their flexibility, they can detect and exploit interactions in data that are invisible to any existing financial theory (Heaton et al., 2016). DL models will be introduced in detail chapter 4.

A key benefit of DL models over GA approaches is that they are technical indicator free. GA approaches utilize human selected features picked from a large set of existing technical indicators. On the contrary, DL approaches rely on automatic feature learning. This means that they are able to utilize the raw price data and find completely unforeseen patterns. This is the main reason we chose DL models over GA approaches in this study.

Due to the novelty of DL technology, there is very limited research available regarding portfolio management. Heaton, Polson and Witte opened the discussion in 2016 with their paper “Deep learning for finance: deep portfolios”. In their work they utilize autoencoders, an ANN based dimensionality reduction technique to denoise equity data of biotechnology companies belonging to the NASDAQ Biotechnology index. They try to beat the index by 1% with mixed results. (Heaton et al., 2016)

Interestingly, a paper from Jiang, Xu and Liang from 2017 compares different deep learning methods in the portfolio management context with cryptocurrencies. In their study they

utilize Bitcoin as a cash asset: they manage a portfolio of cryptocurrencies and try to beat Bitcoin. Their study compares Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) and their goal is to manage a cryptocurrency portfolio in a high frequency context (30 minutes). (Jiang et al., 2017)

The training process of Jiang et al. is the following. All the cryptocurrencies train the same neural network. In other words they tune the same parameters. However, predictions for each cryptocurrency are made individually. Each prediction is based on the high, low and closing prices from 50 previous data points. To take into account transaction costs (0.2% per transaction), the predictions are scored based on the weights of the previous period by utilizing a reinforcement learning agent. (Reinforcement learning will be introduced in a later chapter.) Lastly, the scores from the last step are converted to weights with a softmax function that ensures that the weights of the portfolio assets sum up to one. (Jiang et al., 2017).

The results of Jiang, Xu, and Liang are suspiciously spectacular. They claim that their best model was able to reach at least 4-fold returns in just 50 days. This result cannot be explained by the magnificent bull run cryptocurrencies experienced on the year 2017 since they used Bitcoin as a cash asset. Furthermore, their test set ranged from 9/2016 to 4/2017. (Jiang et al., 2017) This thesis will explore the DL models presented by Jiang, Xu, and Liang further on different market conditions and find whether these results can be replicated and maintained.

A similar architecture to Jiang et al. was implemented by Deng et al. (2016). The authors also combined deep learning with reinforcement learning. Deng et al. implemented their approach on stock-indexes and commodity futures with promising results. Their results indicated that the model is particularly suitable when applied to a trending market condition.. However, their model only traded a single asset and therefore is not a fully-fledged portfolio optimization model.

A key difference between Deng's and Jiang et al.'s approaches was that Deng utilized fuzzy clustering on the raw price data. This basically means reducing the uncertainty of input data by discretizing raw returns to fuzzy linguistic clusters. (For example returns between 4.5% to 5.5% could be discretized to "very high".) The benefit of this technique is that the input vector becomes more robust and less sensitive to small changes. Without fuzzifying, a pure DL model tries to distinguish and find meaning between very close returns (such as +5.23%

and 5.24%). In other words, fuzzifying takes the average of similar returns and teaches the network to find patterns from the robust fuzzified representations. Deng et al.'s fuzzified approach will not be studied further in this thesis. However, we consider it a worthwhile avenue of investigation which could potentially lead to better results than the ones achieved in this study.

3.3. Fundamentals as cryptocurrency price drivers

While this study focuses on finding technical patterns for portfolio optimization, it is also interesting to consider whether there are fundamental price drivers for cryptocurrency prices. Bhambhwani et al. (2019) found evidence that the prices of five prominent cryptocurrencies (Bitcoin, Ethereum, Litecoin, Monero, and Dash) are influenced by two fundamentals: the computing power donated to the network and the adoption level. These two factors therefore provide intrinsic value for mineable cryptocurrencies.

The computing power consists of hash power that miners donated to the cryptocurrency network. The miners donate huge amounts of electricity to the network by solving cryptographic puzzles. While this can be seen as an irresponsible usage of natural resources, it also ensures that transactions flow securely and efficiently. This increases trust of the network as it makes it harder to compromise. However, since the miners are incentivized by block rewards, this fundamental factor is highly endogenous to the price of the cryptocurrency. Furthermore, not all major cryptocurrencies are mineable (for example Ripple).

Bhambhwani et al. (2019) measured adoption level by the number of active addresses used to transact on the blockchain. This is important, as it measures the transaction volume of a given cryptocurrency. Due to network effect, a cryptocurrency with a high amount of transactions is more lucrative.

The problem with these two fundamental factors is that they both are non-stationary. Therefore Bhambhwani et al. (2019) used dynamic least squares to estimate cointegration relationship between the fundamental factors and the prices. This method is also consistent with endogenous variables. The study found that these cointegration terms are statistically significant in longer periods and also in some shorter periods.

Bhambhwani et al. (2019) explained the short-term deviation from fundamental values with two short-term factors: the price of Bitcoin and momentum. Bitcoin's price is the most sentiment driven cryptocurrency as it is the most well-known one. The investor sentiment of Bitcoin also flows to other cryptocurrencies. The study also found evidence that the momentum factor causes a cryptocurrency value to deviate from its fundamental value. Both of these short-term factors were also found statistically significant.

Bhambhwani et al. (2019) study provided concrete evidence that cryptocurrency prices are driven by two fundamentals: computing power and adoption level. However, there are other unobservable fundamentals that affect the valuation. For example, there is an intrinsic value in the technology: cryptocurrencies enable fast anonymous global transactions that are cryptographically secured and do not rely on a third party.

4. Methodology

We are going to implement a trading agent that tries to beat a benchmark index, in our case an equal weighted portfolio of cryptocurrencies. The agent takes as an input raw price data. The task of the agent is to analyze the data and output portfolio weights for the next period. For example, the agent might manage three assets: XRP, ETH and XMR. For each of the cryptocurrency, it has a look in its recent price history and outputs the predicted optimal weight for the next period.

What happens in between the inputs (raw price data) and output (portfolio weights) is implemented with a deep reinforcement learning scheme (DRL). The DRL scheme consists of two components. The first component predicts how each individual asset's price will move on the next period; that is will it increase or decrease in value. The second component takes as an input the information from the first part and decides optimal portfolio weights for the next period. The overall goal will be to maximize expected risk-adjusted return while taking into account transaction costs. The first part is implemented with a deep learning network; more specifically a convolutional neural network (CNN). The second part will be implemented with reinforcement learning; more specifically with policy gradients.

The agent has two ways to maximize expected return. The first way is to choose the initial weights for the backtest period. It is important to choose proper initial weights since large deviations from these will incur large transaction costs. The second way is to rebalance the portfolio by making optimal trading actions.

While the initial weights determine most of the value generated or destroyed during a backtest period, there is still some value to be made (or destroyed) with readjustments. To evaluate the significance between these two components, we implement two agents: the *static agent* and the *dynamic agent*. The static agent only does the first step of choosing initial weights and holds those weights throughout the whole period. The dynamic agent readjusts itself periodically, i.e.: makes trades.

In this chapter, we will first discuss the different strategies (i.e. the agents) that will be evaluated in this thesis. Secondly, we will define the metrics we will use to evaluate portfolio performance. After that, we will introduce deep learning and justify why it is interesting in portfolio management. Thirdly, we will discuss how we used Jiang et al.'s portfolio management framework in our methodology. Finally, we will discuss our deep

reinforcement learning (DRL) framework. This is the framework that will be tested in chapter 5. It basically describes the process of how raw price data is converted in to "optimal" weights.

4.1. Trading strategies

We compare three strategies in this study: the equal weighted, the dynamic agent and the static agent.

4.1.1. Equal weighted

The equal weighted strategy is very simple. For each backtest, the equal weighted strategy starts with equal weights for each asset including the cash asset (Bitcoin). For example, we might manage three assets: XRP, ETH and XMR. The equal weight would place a 25% weight on each of the assets plus an 25% weight on BTC.

Please note that the assets in the equal weighted portfolio are not weighted by anything: volume, market capitalization or by any other metric. It turns out that for this study the pure equal weights are sufficient. As we will show in experiments, the agents will nearly converge to equal weights. This makes the agents' performance comparable to the equal weighted strategy. Nevertheless weighting the benchmark portfolio by some metric could be very useful in a practical setting and enable a broader analysis.

As we will show later, the static agent actually performs worse than the equal weighted strategy. This implies that our agents are worse in choosing initial weights than the simple equal weighted strategy. Therefore the simple equal weight strategy serves as a decent lower limit of weight initialization that the agents fail to meet.

In this strategy we will compare the performance of our trading agents on multiple trading period lengths from 15 minutes to 1 day. The trading period length determines how often the dynamic agent readjusts itself. Since the equal weighted strategy will not do any trading, it's expected portfolio value will stay constant across different period lengths. However, we needed to choose a some period length for measuring its volatility and maximum drawdown. We use the 2 hour price history for these purposes.

4.1.2. Static agent

The static agent is the more simple version of the deep learning agents. The goal of the static agent is to simply choose some weights for a trading period and hold them. It performs very similarly to the equal weighted strategy. In practice, it has a look at the price history just before the backtest and chooses weight allocations based on that. The amount of price history it has a look at is controlled by a *window length* parameter.

The purpose of the static agent is to separate the value added component of the DRL agent's weight initialization. Its difference compared to the equal weighted strategy is purely based on the DRL agent's ability to identify optimal starting portfolio weights for the backtest period. Furthermore, it separates this initialization component from the trading action component. Therefore it functions as a benchmark for the *dynamic agent.*

4.1.3. Dynamic agent

The dynamic agent uses the same initial weights as the static agent and pursues to change these weights periodically in order to maximize return. The difference in the performance between the two agents comes purely from trading action. In other words, it separates the trading action component from the initialization component.

As we will show in the experiments chapter, the trading action performed by the dynamic agent can create value by increasing returns and/ or reducing volatility in specific market conditions. Briefly, it creates value when the assets under management oscillate a lot in a mean reverting fashion. However, it loses value when there has been a fundamental change in an assets' valuation that lasts longer than the backtest period. We will show that it might work as a useful tool in risk management in specific circumstances.

4.2. Cash asset and Metrics

Before discussing our metrics, we will first discuss choosing our cash asset. This choice directly affects our metrics. We considered using USD as a cash asset but chose to use Bitcoin instead. Therefore the performance of our strategies are compared to the price of Bitcoin. This is a good place to explain why.

We wanted to completely isolate our analysis on the cryptocurrency space and did not want to take into account any real money such as the USD. The reason for this choice is two-

folded. Firstly, only few major cryptocurrencies (such as BTC, ETH and XRP) are traded against USD with sufficient volumes. They are usually quoted against Bitcoin. Secondly, there is already a lot of volatility in the cryptocurrency market alone. We wanted to separate this volatility from BTC/USD price action.

Consequently, we use Bitcoin as the risk-free rate. While *we certainly do not by any means consider Bitcoin as a risk-free investment*, we wanted to isolate volatility only to the cryptocurrency dimension. While *Bitcoin certainly is a high risk investment*, it is considerably less riskier than the other cryptocurrencies.

Taking Bitcoin as our cash asset, we use three performance measures to evaluate the performance of our agents: accumulated portfolio value, maximum drawdown and Sharpe ratio. These were the same metrics used by Jiang et al. (2017).

Firstly, we use accumulated portfolio value, which simply compares the final value of the portfolio to the initial investment. It measures how much value the strategy created or destroyed during the period compared to the cash asset.

$$p = p_t / p_0$$

In all our backtests we assume that the initial investment, p_0 is one cash unit. Therefore the equation above just simplifies to:

$$p = p_t$$

We used the traditional Sharpe ratio to estimate the risk-adjusted return of our portfolio. It is measured as the excess return of the portfolio per a unit of volatility:

$$S = \frac{E(p_t - p_f)}{\sqrt{var(p_t - p_f)}}$$

Since we use Bitcoin as a cash asset and risk-free rate, its value is by definition constantly one and therefore its standard deviation is zero. Therefore the above Sharpe ratio simplifies to:

$$S = \frac{E(p_t)}{\sqrt{var(p_t)}}$$

We considered annualizing the Sharpe ratio. The annualized version is actually reported in some metrics of this study. However, it is not very meaningful since it just multiplies the

ratio with a constant (~2.7). Since we are using the same period (50 days) in each of our backtests, the vanilla Sharpe ratios are already directly comparable. Moreover, since the returns in cryptocurrency markets are so volatile, the Sharpe ratios are already highly inflated when compared to traditional assets. Finally, according to a paper by Lo (2003) the whole process of annualizing Sharpe ratios works only in special circumstances. In most cases it does not take properly into account serial correlation and the fact that both expected return and volatility are estimated unknown quantities.

Nevertheless, we used the following formula to annualize our Sharpe ratios:

$$S_{ann} = S * \sqrt{\frac{365}{50}}$$

We used 365 days instead of the traditional 252 trading days since cryptocurrencies are traded all year round.

A final measure we used was the maximum drawdown (MDD). Similar to Sharpe ratio, it also takes into account the volatility of our portfolios. Since the Sharpe ratio uses standard deviation to measure risk, it considers both upward and downward price action as “risky”. Contrarily, the maximum drawdown only considers downside risk by measuring the maximum loss of value during the investment period. It is defined as the peak value minus the trough value divided by the peak value:

$$MDD = max_{t,u \in (0,T)} \frac{p_t - p_u}{p_t}$$

In other words, it measures how large proportion of the investment was lost when comparing the peak value to the lowest point.

4.3. Deep learning

Before diving deeper into our model, we first need to understand how deep learning works on a conceptual level. We will first discuss deep learning generally and provide an intuition on why it is so powerful in many modern applications. Secondly we will discuss how it might be useful in portfolio optimization.

4.3.1. Deep learning: an intuition

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

"Cells that fire together wire together."

Donald O. Hebb, Father of neuropsychology and neural networks

Deep learning (DL) models are machine learning models that are *loosely* inspired by the architecture and functions of human brains. They share the same fundamental idea of a large connected system that is based on simple processing units called artificial neurons. The neurons interact with one another on a massive scale by transmitting simple signals. While each neuron is very simple, intelligent behavior arises as an emergent property from complex interactions.

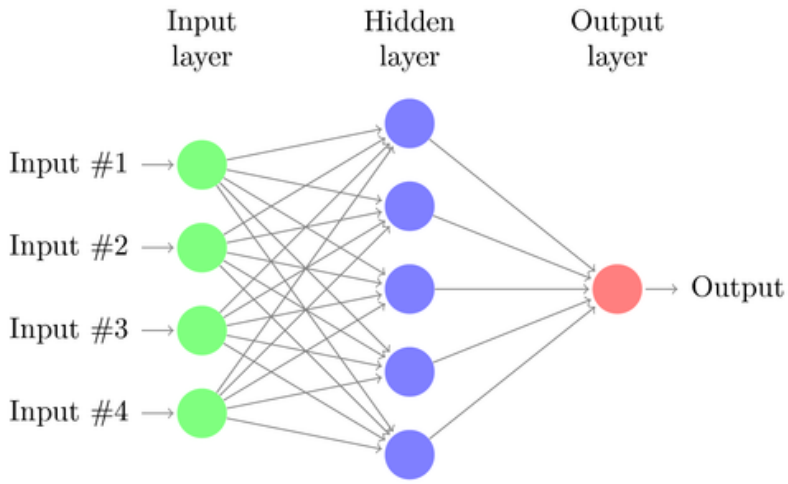


Figure 2: A simple artificial neural network (MLP)

The goal of deep learning is to implement an artificial neural network (ANN) that solves some arbitrary statistical prediction problem. Even the most simple deep learning architecture, the multilayer perceptron (MLP) turns out to be very powerful in expectation (Figure 2). According to the Universal approximation theorem, an MLP with just a single hidden layer can approximate any continuous function mapping from one finite dimensional discrete space to another (Hornik 1991, Cybenko 1989). In practice, this means that it can

approximate any linear or nonlinear function. What the theorem does not consider is whether the MLP is actually *able to learn a good approximation*. The approximation is found by iteratively training the MLP and will be discussed in appendices.

ANNs commonly have at least hundreds of neurons and they generally work better when the amount of layers and neurons are increases. However, training a larger neural network takes significantly more computational time. The added value of extra neurons and/ or layers of neurons gradually converges to zero. Furthermore, after a huge ANN has been trained, most weights between neurons have been adjusted to near zero values. Consequently, each neuron works kind of like a sniper: a neuron only fires on very specialized cases when specific neurons have sent it a signal.

Deep learning models break an arbitrary prediction problem into a hierarchy of concepts. Ian Goodfellow, a leading deep learning researcher, encapsulates this well in his book "Deep learning": "[Deep learning models] allow computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined in terms of its relation to simpler concepts. By gathering knowledge from experience, this approach avoids the need for human operators to formally specify all of the knowledge that the computer needs. The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to AI deep learning." (Goodfellow, 2016)

DL methods outperform other machine learning models when there is a lot of data available and the task is particularly hard for computers. More specifically, DL is able to solve problems that humans solve intuitively but which are hard to describe formally. Some examples include recognizing spoken word or categorizing images. (Goodfellow, 2016) Deep learning models are nowadays used in a wide range everyday applications. For example, many intelligent Google products such as Google Search, Google Translate and YouTube video recommendations utilize DL models. Furthermore, DL technology is used in practically every recommendation engine from Netflix to Amazon.

4.3.2. Deep learning: motivation for portfolio management

The powerful implications of the universal approximation theorem are also interesting from a portfolio management perspective. Since ANNs can approximate an arbitrary nonlinear

function, they are capable of finding patterns in data without explicitly being told what to find. Due to their flexibility, they can detect and exploit interactions in data that are invisible to any existing financial theory (Heaton, Polson, Witte 2016). If there exists a function exists that maps previous price history to future prices, the universal approximation theorem guarantees that an ANN exists that approximates that function.

Another key benefit of DL models in portfolio management is that they do not require feature engineering. An financial example of feature engineering would be to convert raw price data to technical indicators such as Bollinger bands. The performance of other popular machine learning models such as Support Vector Machines (Corinna, Vapnik 1995) is usually greatly dependent on the specific transformation of the raw data they are fed as input. The process of manually designing good features requires a lot of human effort. Contrarily, DL methods do not require this and work better when they also learn the representation itself (Goodfellow, 2016).

The most simple DL network for portfolio optimization could be the following. In the input layer we have a neuron for each asset we want to manage. For example, say that we want to manage three assets: XMR, ETH and XRP. In this particular example the input layer would have three neurons; one for each asset. Similarly, the output layer would have three neurons: the weights for each asset in the next period. The data type in the input neurons would be a vector of price history. In other words, for each asset we would like to manage, we feed the neural network the relevant price history in a vectorized format. The output neurons that hold the weights could be simple floating point numbers that sum up to unity.

However, this simple model has two key weaknesses. Firstly, it does not take into account the previous period weights at all. This is necessary in order to take into account transaction costs. Secondly, it does not take as an input the performance of its previous actions. Therefore it has no way of knowing whether it made a good or bad trading decision on the previous period. To mitigate these issues, we introduce a reinforcement learning component. The reinforcement learning component will be discussed later.

This amount of deep learning knowledge should suffice in understanding this study. As the experiments will demonstrate, the deep learning model converges to a relatively traditional equally weighted strategy. This strategy could be easily mimicked with a more simple algorithm. Unfortunately, this makes the DL part of our model somewhat redundant.

Consequently, we do not consider it necessary for the reader to understand deep learning on a deeper level. However, if you wish to learn more about deep learning, we included an entry level package on the appendices that shows why deep learning is such an important technology right now. It also demonstrates how a deep learning network makes predictions and more importantly how it is trained by exploiting the chain rule of calculus.

4.4. Jiang et al.: DRL framework for portfolio optimization

This thesis is heavily influenced by the paper by Jiang, Xu and Liang called "A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem" (Jiang et al., 2017). We selected their best performing architecture (a convolutional neural network) and introduced a few modifications into it. We also extended their analysis to different trading periods and to more recent data.

According to our knowledge, Jiang et al. introduced the first deep reinforcement learning framework for the financial portfolio management problem. This framework can be utilized in high-frequency algorithmic trading. Since the framework is model-free, it is in principle agnostic to the actual assets traded and could be applied to any markets. (Jiang et al., 2017)

While there are previous attempts that utilize deep reinforcement learning for the algorithmic trading problem (e.g.: Deng et al 2017), Jiang et al.'s paper is the first one that is not limited to single-asset trading. Furthermore, many machine learning approaches have focused on identifying trends and predicting price movements (e.g.: Heaton, 2016). Contrarily, Jiang et al. take it a step further and try to directly optimize trading actions that maximize expected return. (Jiang et al., 2017).

The authors used semi-high frequency cryptocurrency data. Their portfolio consist of a cash asset (Bitcoin) and 11 other most-volumed cryptocurrencies. Trading actions are made every 30 minutes. They compare the trading agent's performance to well-established trading algorithms with three performance metrics: Sharpe ratio, Accumulated Portfolio Value and Maximum drawdown. The authors claim that their trading agents outperform all the other trading algorithms and are able to achieve 4-fold returns in 50 days. (Jiang et al., 2017)

The framework we are going to present next is based on the work of Jiang et al.

4.5. Deep reinforcement learning framework

In this section we will introduce the deep learning framework. Previously we discussed that we want to have a trading agent that takes as an input some raw price data and outputs optimal weights for the following period. We have yet to discuss how the raw price data is converted to those optimal weights. This section concentrates on that process.

We will implement a deep reinforcement learning (DRL) model for this problem. The DRL model consists of two components: firstly deep learning and secondly reinforcement learning. The goal of the deep learning component is to extract useful information from raw price data and convert it to a meaningful representation. The reinforcement learning component then takes this optimal representation and figures out how to rebalance the portfolio while taking into account transaction costs.

We will begin by briefly discussing reinforcement learning and justify why it is an necessary component for our framework. We will introduce the protagonist of this study: the reinforcement learning agent. Secondly, we will formalize our optimization goal. This is what the agent seeks to maximize. Thirdly, we will discuss the representation of the problem we are giving to the agent (state) and the way that the agent interacts with its environment (actions). Fourthly, we will discuss how the agent makes decisions by discussing our deep learning policy. Fifthly, we will discuss epsilon greedy exploration; a technique we used to encourage the agent to explore a large portion of the search space. Finally, we will present our hyperparameters. Hyperparameters are the specific settings and configurations that control our deep learning policy.

4.5.1. Reinforcement learning in portfolio optimization

We want to implement an intelligent agent that periodically optimizes a cryptocurrency portfolio. The first step is to come up with an way to find meaningful signals from noisy financial data. In other words, we want to be able to predict asset prices for the next period based on historical data. For this problem we use deep learning. However, as pointed out by Jiang, et al. (2017) additional logic needs to be added to convert these predicted prices to market actions. For this problem we will utilize reinforcement learning.

Reinforcement learning (RL) is a field of machine learning that studies how agents learn by trial and error. The key components of an RL model are the **environment** and the **agent**.

The goal of the agent is to interact with the environment via actions. For each action, the agent receives a reward or a penalty. The performance of the agent is determined by the cumulative *rewards* it receives from multiple *actions*. The environment can be complex and stochastic. The core idea of RL is very general and it can be applied to a wide range of tasks where the task is to perform a sequence of actions.

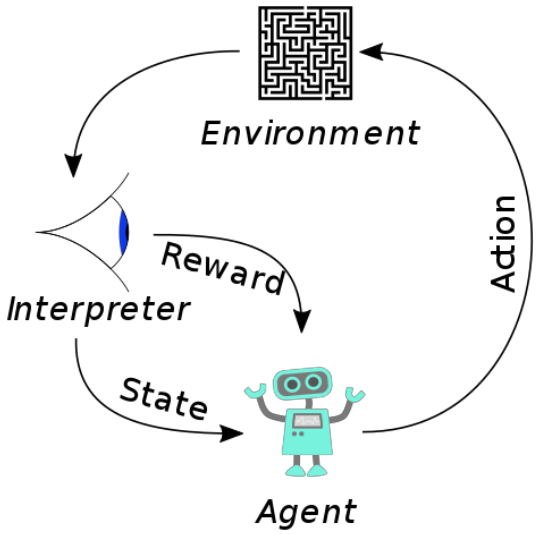


Figure 3: The basic loop of reinforcement learning

Let's have a look at the picture above (Figure 3). In our case the agent is our algorithmic portfolio manager. The environment is the cryptocurrency market. The agent interacts with the cryptocurrency market (environment) by choosing asset weights (actions). After each action, there is some change in the market as prices fluctuate. The new market condition is fed to an interpreter. The interpreter calculates the change in portfolio value (reward). It also gathers the relevant new price data from the environment and a snapshot of the previous periods weights (state). The new price data/ previous weights (state) and the change in portfolio value (reward) are passed back to the agent. With this information, the agent uses a *deep learning policy* to determine a new action.

More formally, the goal of an reinforcement learning agent is to maximize the cumulative expected reward *U,* it expects to receive from a sequence of actions. Each individual reward can be denoted as:

$$r_t = R(s_t, a_t)$$

, where s_t is the current state of the world and a_t some action just made. In other words, the individual reward is a function of some action taken on some state of world.

We want to maximize the cumulative return, *U,* of a sequence of action state pairs:

$$U = R(s_1, a_1, s_2, a_2, \ldots, s_t)$$

We will next define the left side of the above equation, the cumulative reward in our portfolio optimization case. After that we will discuss the right side of the equation; define the states and actions.

4.5.2. Optimization goal: cumulative and periodic rewards

Next we are going to define the rewards of our agents. This is important since, the reward is the actual optimization goal we wish to maximize. We are going to define our reward similarly as Jiang et al. did in their study. The cumulative reward of our agent is the expected logarithmic cumulative return:

$$R(s_1, a_1, \ldots, s_t, a_t, s_{t+1}) = \frac{1}{t_f} ln \frac{p_f}{p_0}$$

, where t_f is the amount of trading periods, p_f is the final portfolio value and p_0 is the initial portfolio value.

However, this notation is not enough as it does not tell us the rewards of a single period. Therefore we decompose the total change in portfolio value to the sum of individual rewards gained at each period:

$$ln \frac{p_f}{p_0} = \sum_{t=1}^{t_f+1} r_t$$

, where r_t is the logarithmic return of the portfolio for a single period.

The logarithmic return for a single period is further defined as:

$$r_t = ln(y_t \times w_{t-1} \times \mu_t)$$

, where y_t is a relative price vector that describes how each asset's price has change during period *t,* w_{t-1} are the previous periods weights and μ_t is a transaction remainder which encapsulates how transactions costs dilute the portfolio value.

4.5.3. States and actions: interacting in the financial environment

Below we are going to define the states and actions of our agents. These are also important: state consists of the representation on how the agent understands the world. Actions on the other hand consist of how the agent interacts with the world.

We are going to define our states and actions the same way as Jiang et al. did in their study. Actions will consist of portfolio weights for each period. For example an action *a* at time period *t* is defined as:

$$a_t = w_t = (w_{1,t}, w_{2,t}, \ldots, w_{n,t})$$

In each time period we have a state object s_t. It consists of two components, the previous periods weights and a price tensor:

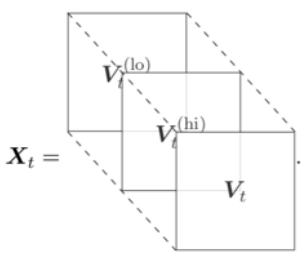
$$s_t = (w_{t-1}, X_t)$$

, where w_{t-1} is simply the previous period's weights. These are necessary to take into account to minimize transaction costs in the cumulative reward.

$\boldsymbol{X_t}$ is a price tensor consisting of raw price data. In other words, it is a data structure that holds historic price data. The price tensor has a rank of 3 and a shape of *(f, n, m):*

- *f:* The number of features. Similar to Jiang et al. (2017), we use three features (high, low, and closing prices)
- *n:* The number of periods. We used 70 periods
- *m:* The number of assets traded by the agent (11 + Bitcoin)

Since only the changes in prices are relevant for the performance of the trading agent, the prices are normalized by the latest closing price (Figure 4):



$$\boldsymbol{V}_t = \left[\boldsymbol{v}_{t-n+1} \oslash \boldsymbol{v}_t \middle| \boldsymbol{v}_{t-n+2} \oslash \boldsymbol{v}_t \middle| \cdots \middle| \boldsymbol{v}_{t-1} \oslash \boldsymbol{v}_t \middle| \mathbf{1}\right],$$

$$\boldsymbol{V}_t^{(\mathrm{hi})} = \left[\boldsymbol{v}_{t-n+1}^{(\mathrm{hi})} \oslash \boldsymbol{v}_t \middle| \boldsymbol{v}_{t-n+2}^{(\mathrm{hi})} \oslash \boldsymbol{v}_t \middle| \cdots \middle| \boldsymbol{v}_{t-1}^{(\mathrm{hi})} \oslash \boldsymbol{v}_t \middle| \boldsymbol{v}_t^{(\mathrm{hi})} \oslash \boldsymbol{v}_t\right],$$

$$\boldsymbol{V}_t^{(\mathrm{lo})} = \left[\boldsymbol{v}_{t-n+1}^{(\mathrm{lo})} \oslash \boldsymbol{v}_t \middle| \boldsymbol{v}_{t-n+2}^{(\mathrm{lo})} \oslash \boldsymbol{v}_t \middle| \cdots \middle| \boldsymbol{v}_{t-1}^{(\mathrm{lo})} \oslash \boldsymbol{v}_t \middle| \boldsymbol{v}_t^{(\mathrm{lo})} \oslash \boldsymbol{v}_t\right],$$

Figure 4: Price tensor (from Jiang et al., 2017)

, where each V_t consists of normalized prices and the $\oslash$ is an element-wise division operator.

At the end of each period, the trading agent computes a new weight vector w_t by using the price tensor X_t and the previous weights w_{t-1}. Therefore the new weights w_t can be seen as an action performed by the agent:

$$a_t = w_t = \pi(X_t, w_{t-1})$$

, where π is the agents policy.

A policy is an arbitrary algorithm that select actions. In our case we are using a deep learning policy. In the next section we will define the deep learning policy used in this study.

4.5.4. Choosing the optimal action with a deep learning policy

Here we will define the deep learning policy used in this study. In the previous sections we discussed what we pursue to maximize: the cumulative reward. We also discussed state which describes how our agents observe the world. Finally we discussed actions which define how the agent can interact with its world. Yet one key component has yet to be discussed. We have not yet defined how the agent is going to transform its observation of the world into an action. Next we are going to discuss our policy which does exactly that.

We used a deep learning policy to determine actions. However, as the experiments will show, the deep learning policy was not particularly exciting as it converged nearly to an equal weighted strategy. Therefore we do not consider it meaningful to describe the literature behind its architecture in detail as it could easily be replaced with a simpler version. Likewise, it is not relevant to be able to fully understand the architecture to be able to interpret the result of this study. If you wish to learn more about convolutional neural networks, the softmax function and other key deep learning concepts, please refer to the appendices. *Without reading these, this chapter will be quite difficult to follow.*

While the details of the neural network implemented are beyond the scope of this study, one key point is important: we used the same neural network for each asset. This is important since the neural network cannot learn to distinguish between assets. It cannot for example learn to remember that some particular asset, say XRP had a great bull run far back in history

and give it more weight. Rather it is forced to look at each asset just the same and make democratic trading decisions.

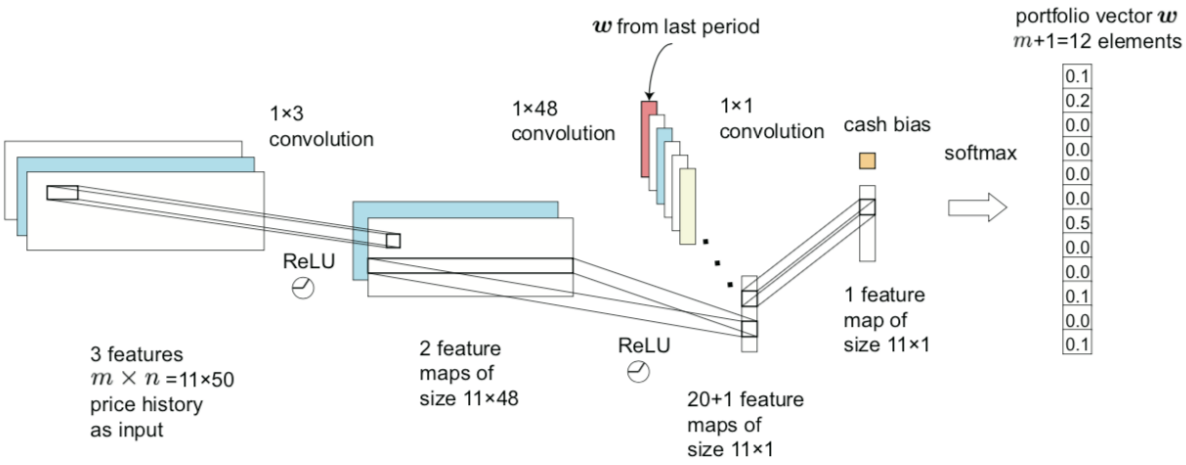


Figure 5: Convolutional policy network (from Jiang et al., 2017)

Next we are going to describe the deep learning policy layer by layer. Our policy function is a convolution neural network that takes as an input the price tensor, $\boldsymbol{X_t}$, we defined in the previous sector. The price tensor had a rank of three and consisted of 1) the number of assets 2) number of features (high, low and close) and 3) the number of time periods (in our case 70).

The picture above (Figure 5) summarizes our CNN architecture. The input tensor is fed into a convolution layer that reduces its dimensionality. This is useful as it reduces the amount of computation required to analyze the input data while keeping relevant information. The output tensor is then fed to a second convolutional layer that reduces the dimensionality even further.

After the convolutional layers, we add the weights from the previous period. Remember that we defined our actions as:

$$a_t = w_t = \pi(X_t, w_{t-1})$$

We already fed our price tensor, $\boldsymbol{X_t}$ to the convolutional layers. However, we input the previous weights w_{t-1} after these layers. The convolutional layers are designed to find meaningful patterns from some noisy environment which is not in our control. It doesn't make much sense to try to find signals from the weight vector that we control.

We stack the previous weights and the output tensor of the convolutional layers into a one dimensional vector. To take into account our cash asset, we also include that to the vector. Then we feed this vector into a regular fully connected deep learning layer. Finally, the outputs of this layer are fed to a softmax function that ensures that the final weights sum up to one. The softmax function outputs a normalized vector which is the weight vector for the next period.

4.5.5. Epsilon greedy exploration

Above we discussed our deep learning policy, the algorithm used by the agent to make actions. However a problem with the algorithm is that it gets less and less random through time as the network starts to converge into some local optima. A common way to mitigate this problem is to use epsilon greedy exploration: once in a while we ignore the action given by the network and make a random action. This ensures that the policy continues to explore new ways to gain higher rewards instead of being content with what it has found.

In practice, we choose some threshold ε. At each period the policy acts randomly with probability ε and greedily by choosing the optimal value with probability $1-\varepsilon$. Consequently, the algorithm will spend more time exploring the unknown. This is naturally only done in the training phase. We utilized this while training our networks.

4.5.6. Hyperparameters

Deep learning networks are massive architectures that consist of many parameters you can tweak and tune. These parameters are called hyperparameters. Below is a list of the hyperparameters used in this study. These are important if someone wants to replicate this study.:

- **Window length:** 70 (the amount of periods the agent has access to when considering trade actions)
- **Number of filters in convolutional layer 1:** 5 (the amount of different patterns the convolutional layers seek to find)
- **Number of filters in convolutional layer 2:** 50
- **Kernel size:** 3 (the size of patterns the convolutional layers are looking for)
- **Trading cost:** 0.2% (the highest cost collected by our data source, Poloniex)
- **Epsilon greedy threshold:** 0.8 (policy took a random action with 20% probability)

- **No. of episodes:** 3 (How many times the training data was gone through in the training phase)
- **Batch size:** 50 (How often the neural network is updated in each episode)

Often in machine learning projects, some part of the dataset is left as a validation set which is used to tune the hyperparameters of the model (such as the sizes of the neural network layers). However, we did not do any hyperparameter optimization since we noticed that our model was quite insensitive to hyperparameter changes. We therefore manually settled with these suitable hyperparameters.

5. Experiments and results

In this chapter we present the results of our study. Turns out that our deep reinforcement learning agents converge to follow a quite boring strategy: an equally weighted strategy that balances itself periodically. However, the agents turn out to be useful in some market conditions.

The structure of this chapter is the follow. First, we explain the strategy followed by the dynamic agent with a simplified example. The example will demonstrate how the agent works.

Secondly, we introduce our backtest periods. The backtest periods were chosen based on uniquely interesting price action.

Thirdly, we will discuss how we selected the tradable assets for our portfolios and our data source, Poloniex.

Fourthly, we will show how we improved the robustness of our results by running numerous simulations.

Fifthly, we will discuss our report types.

Then, we will go through the individual backtests. We will analyze the different agents performance in the backtest periods and explain what worked and what did not.

Finally, we will wrap the backtests up by presenting concluding statistics and also discuss which trading period length worked the best.

5.1. Dynamic agent's strategy: Auto-balancing equal weight

Before going into the individual backtests, we will first describe the strategy that the dynamic agent follows across all backtests. It turns out, that the agent follows the same strategy across all back tests. Each backtests starts with the agent holding 100% of its value in the cash asset (Bitcoin) and it immediately balances itself to follow an equally weighted strategy. We now present a simplified backtest example (Figure 6) where only three coins plus Bitcoin (cash asset) are traded.

During our example backtest, the cryptocurrency Ripple (XRP) had a bull run were it over 10-folded its value. This is indicated by the blue line in the “Cryptocurrency price evolution”-chart. Please note that in the “Cryptocurrency price evolution”-chart, each coin is scaled to start from the value 1 in the beginning of the period. The y-axis is logarithmic to enhance readability.

The “Portfolio weight evolution”-chart shows how the dynamic agent rebalances itself during the period. In this particular example the blue dots indicate Ripple’s (XRP) weight. We can see that when Ripple’s value sharply increases during 2017-03-29 to 2017-04-05, the dynamic agent sells majority of its position to once again arrive to equal weights.

In the “Portfolio value”-chart we can evaluate the agent’s performance. During the volatile week of 2017-03-29 to 2017-04-05 an equal weighted strategy outperforms the dynamic strategy as Ripple continues to increase in value. However, as the price of Ripple eventually drops, the value of dynamic agent comes close to the equal weighted value.

Further, we can see from the table on the top left corner that the dynamic agent has a higher Sharpe ratio and lower maximum drawdown. While it has a lower return, it also has significantly lower volatility.

Overall, the dynamic agent excels in a market environment where there is a lot of upward and downward price action. It turns out that there are specific market conditions in the cryptocurrency space were this mean reverting strategy is quite robust.

While running simulations we noticed some peculiarities of the agent. It turns out that the initial weights of the agent are not exactly equal and sometimes not even close. We will demonstrate this in the backtest “Ripple bull run”. Furthermore, the agent tends to have a higher cash weight on longer trade periods (4 h and 1 day). The reason for this is unknown. However, due to these two factors, the static agents performance and the equal weight performance can differ from iteration to iteration.

Dynamic agent demo

Strategy	Ptf value	Sharpe	Sharpe (ann.)	MDD
Dynamic agent	2.7128	2.4675	6.6668	0.4659
Static agent	2.8831	2.3166	6.2592	1.7334
Equal weighted	2.8323	2.3208	6.2704	1.6829

Dataset	Start date	End date	Days	Steps
Train period	2017-01-01	2017-03-06	64	393
Test period	2017-03-07	2017-04-27	50	303

Parameter	Value
No. batches	10
No. episodes	1
Batch size	50
Trading fee	0.2%
Trading period	4h
Window length	70
Conv Filters, 1	5
Conv Filters, 2	50
Kernel size	(1, 3)
ε greedy thld	0.8

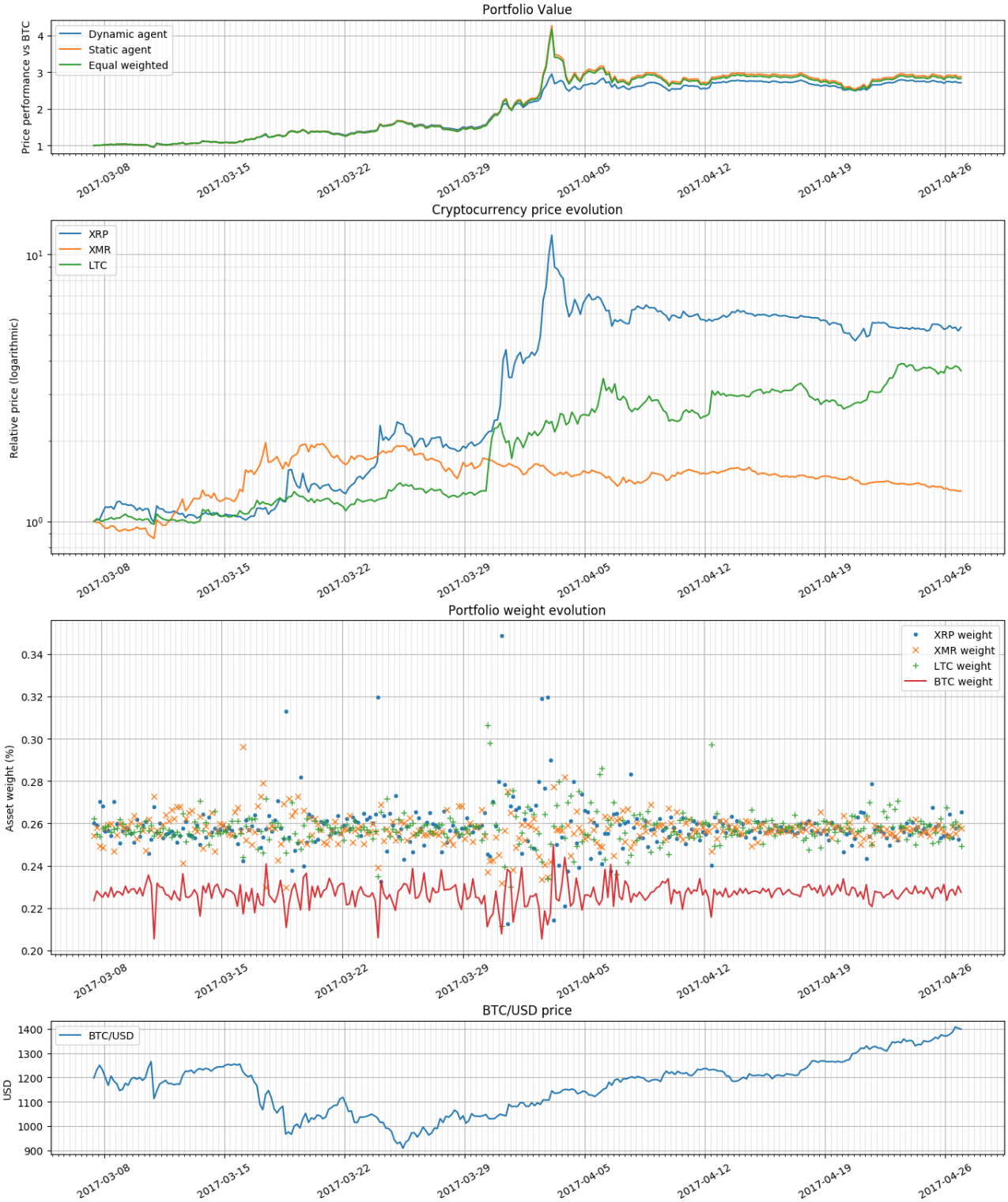


Figure 6: Simulation summary example

5.2. Choosing backtest periods

We ran backtests on seven date ranges. Each backtest period lasted for 50 days. This makes their performance comparable. In each backtest period, we evaluate the agent's performance on five different trading periods: 15 minutes, 30 minutes, 2 hours, 4 hours and 1 day. As a results, we have 35 backtest settings. Each backtest date range meets one or both of the following conditions:

I. There was significant a significant change in Bitcoin's price
II. There was significant turbulence regarding Bitcoin's dominance (i.e.: Bitcoin's percentage share of the total cryptocurrency market cap)

Each backtest period has some uniquely interesting price action which we tried to capture with descriptive names:

1. **Calm before the storm:** After a long recession, the cryptocurrency market starts to display some confidence on Bitcoin.
2. **Awakening:** Bitcoin's price goes above the price of 1000 USD for the first time since December 2013
3. **Ripple bull run:** Bitcoin's dominance goes below 70% for the first time in history due to the seminal bull run of Ripple.
4. **Ethereum valley:** Bitcoin's dominance plummets below 40% as Ethereum nearly becomes the most valuable cryptocurrency in market capitalization.
5. **All-time high:** Bitcoin's price reaches its all-time high of 20 000 USD
6. **Rock bottom:** Bitcoin's price plummets to its lowest level after the all-time high.
7. **Recent:** On April 2019, confidence seems to return on the cryptocurrency market as Bitcoin's price almost doubles.

On the next page we present graphics regarding Bitcoin's price and dominance for the backtest periods. While Bitcoin's price is interesting for this study (Table 1), a greater emphasis should be placed on dominance (Table 2). Since we are using Bitcoin as a cash asset, the change in Bitcoin's dominance directly tells us sign and the magnitude of the Sharpe ratio. Backtests' relation to dominance is visualized on the "Percentage of Total Market Capitalization" chart. (Figure 7)

Table 1: Bitcoin price action summary

			BTC price (USD)					
#	Name	Date range	Start	End	High	Low	End/ Start	High/Low
1	Calm before the storm	2016-09-07 to 2016-10-28	610.57	688.31	688.31	596.03	12.73%	15.48%
2	Awakening	2016-12-08 to 2017-01-28	768.08	919.75	1183.32	760.96	19.75%	55.50%
3	Ripple bullrun	2017-03-07 to 2017-04-27	1273.21	1281.08	1293.47	905.92	0.62%	42.78%
4	Ethereum valley	2017-05-28 to 2017-07-18	2054.08	2228.41	2990.58	1868.12	8.49%	60.09%
5	All-time high	2017-11-23 to 2018-01-13	8077.95	13980.6	19974.1	7986.32	73.07%	150.10%
6	Rock bottom	2018-11-10 to 2018-12-31	6411.76	3865.95	6421.81	3199.01	-39.71%	100.74%
7	Recent	2019-03-06 to 2019-04-26	3913.23	5281.63	5633.01	3870.04	34.97%	45.55%

Table 2: Backtest analysis - Bitcoin dominance summary

			BTC dominance (%)					
#	Name	Date range	Start	End	High	Low	End/ Start	High/Low
1	Calm before the storm	2016-09-07 to 2016-10-28	80.080%	82.190%	82.220%	78.590%	0.021	0.036
2	Awakening	2016-12-08 to 2017-01-28	85.830%	85.270%	88.110%	84.540%	-0.006	0.036
3	Ripple bullrun	2017-03-07 to 2017-04-27	84.480%	65.910%	84.710%	64.520%	-0.186	0.202
4	Ethereum valley	2017-05-28 to 2017-07-18	48.690%	46.470%	49.410%	37.100%	-0.022	0.123
5	All-time high	2017-11-23 to 2018-01-13	56.270%	32.730%	67.460%	32.510%	-0.235	0.350
6	Rock bottom	2018-11-10 to 2018-12-31	52.320%	51.720%	55.330%	50.690%	-0.006	0.046
7	Recent	2019-03-06 to 2019-04-26	51.590%	54.500%	54.900%	50.060%	0.029	0.048

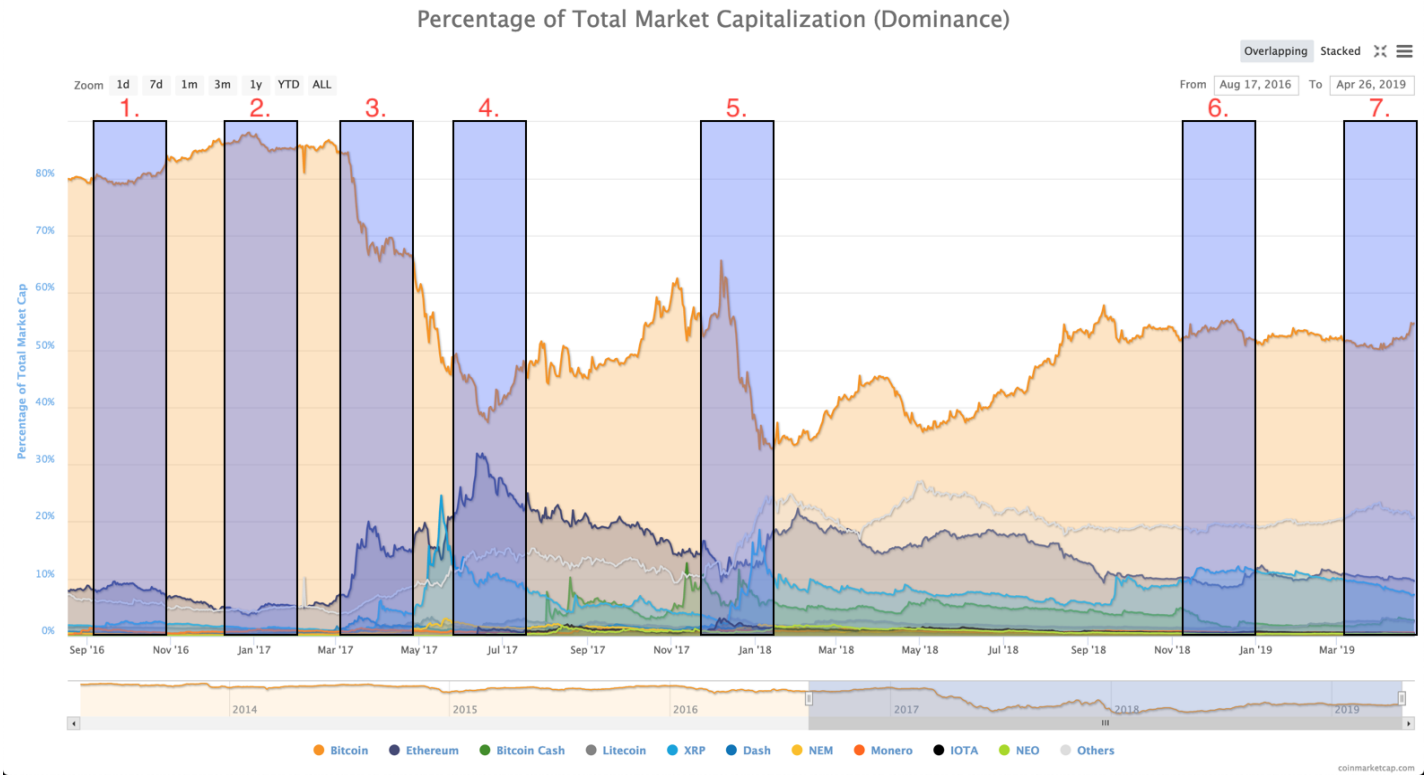


Figure 7: Backtest analysis - Bitcoin dominance evolution (from coinmarketcap.com)

5.3. Dataset and asset selection

The data used in this study was fetched from Poloniex, a cryptocurrency exchange from USA. Poloniex is one of the oldest cryptocurrency exchanges and it was also used by Jiang et al. (2017) as their source for data. Poloniex has a very comprehensive API (Application programming interface) that allows interacting with the exchange programmatically. This allows a trading algorithm to fetch datasets and make market orders very flexibly. Further, Poloniex offers high-frequency data for periods as short as 5 minutes.

Similar to Jiang et al. (2017), we selected our assets by preselecting the coins with the highest 30 day average volume on Poloniex. As Jiang et al. pointed out this was important to be able to meet two important assumptions regarding the trading agent:

1. Zero slippage: The liquidity of the traded assets is so high that each trade can be carried out immediately at the last price when an order is placed.
2. Zero market impact: The capital invested by the trading agent is so insignificant that it has no influence on the market prices.

The volume information is gathered before the backtest period to mitigate survivorship bias. Otherwise future price information could be indirectly passed to the algorithm. 30 day average volume was used instead of daily volume since coins can have sudden boosts or drops in trading volume. (Jiang et al., 2017). The table below (Table 3) displays the assets traded in each backtest period.

Table 3: Cryptocurrency portfolio for each backtest

		Assets										
#	Name	1	2	3	4	5	6	7	8	9	10	11
1	Calm before the storm	XMR	ETH	USDT	DASH	LTC	ETC	FCT	MAID	LSK	BTS	STEEM
2	Awakening	XMR	ETH	USDT	DASH	XRP	ETC	ZEC	FCT	REP	STEEM	MAID
3	Ripple bullrun	XMR	ETH	USDT	DASH	XRP	LTC	ETC	MAID	FCT	GNT	ZEC
4	Ethereum valley	XMR	ETH	USDT	DASH	XRP	LTC	ETC	STR	XEM	DGB	ZEC
5	All-time high	XMR	ETH	USDT	DASH	XRP	LTC	ETC	BCH	STR	VTC	LSK
6	Rock bottom	XMR	ETH	USDT	DASH	XRP	LTC	BCH	STR	BCHSV	ZRX	ZEC
7	Recent	XMR	ETH	USDT	DASH	XRP	LTC	STR	BCHABC	BCHSV	EOS	DGB

Later we will refer to some of the more important cryptocurrencies by their full names. These include Ethereum (ETH) and Ripple (XRP). However, we do not consider it necessary to describe each cryptocurrency used in this study in detail. This is because the assets were purely chosen based on volume. No human judgement was involved in asset selection.

We tried to use as much data as sensibly possible for training the network. For the 15 minute trading period, the network was trained with 249 days of data. For 30 minute and larger trading periods, we utilized 549 days of data. We did not find it purposeful to use older data since we considered it unnecessary to train the network to find patterns from so old data.

5.4. Simulations

Training a deep learning network is a stochastic process and does not guarantee converging into a stable solution. Even if such a solution would exist in expectation, guaranteeing that solution would require impractically heavy computation. Especially deep reinforcement learning (DRL) is notoriously unstable. Therefore our DRL portfolio optimizer might excel (or do very poorly) in a single backtest purely by chance.

To improve the robustness of our results, we ran numerous simulations. In each simulation, we trained the neural network from scratch to ensure that each instance was independent. This was highly computationally expensive, but luckily the authors had access to a Linux supercomputer. Even with high performance resources, the computations took a couple of days to complete. In total we ran 1002 simulations.

The models were built using the Python programming language. More specifically, we used the Tensorflow library which is an open-source deep learning developed by Google Brain. It is the most popular deep learning library and powers many of Google's intelligent products.

5.5. Backtest reports

We offer three different reports for all our backtests. Firstly, the aggregated report that compares the performance across different period lengths. Secondly, the simulation report that shows the performance of a single iteration of simulation. Finally, we offer the stability report that shows how stable the simulations were.

5.5.1. Aggregated report

The aggregated report is the most important of the reports. It gathers together all the simulations we did for a specific backtest and offers cross-sectional analysis for different strategies and period lengths. We provide an example on the next page (Figure 8).

On the table on top we present the portfolio value change, maximum drawdown and Sharpe ratio for the dynamic agent, static agent and equal weighted portfolio for each trading period. We also display the amount of simulations we did for each trading period length. The first three charts display the same data as the table in graphical format.

The two charts below compare the performance of the dynamic agent compared to the static agent. Since they start from the same allocation, the charts effectively capture the value created or destroyed by the trading action.

The Added portfolio value is defined as:

$$p_{added} = \frac{p_{dynamic}}{p_{static}} - 1$$

Similarly, the added standard deviation is defined as:

$$\sigma_{added} = \frac{\sigma_{dynamic}}{\sigma_{static}} - 1$$

The two charts are important in identifying the best rebalancing period. In Figure 8 we can see that the 2 h rebalancing period destroyed the least value (Added Ptf. Value chart on the left) and added the least amount of additional volatility (Added Stdev chart on the right). This pattern were 30 min and 2 h rebalancing periods worked the best will be a recurring theme in the results of this study.

Calm before the storm			Dynamic agent			Static agent			Equal weighted		
#	Period	Simulations	Ptf. value	MDD	Sharpe	Ptf. value	MDD	Sharpe	Ptf. value	MDD	Sharpe
1	15min	21	-29.76%	0.326	-3.378	-26.34%	0.2925	-3.3151	-23.24%	0.2661	-3.2455
2	30min	19	-30.58%	0.331	-3.3736	-27.98%	0.3063	-3.3227	-23.24%	0.2661	-3.2455
3	2h	33	-31.18%	0.3315	-3.4271	-28.87%	0.3077	-3.3634	-23.24%	0.2661	-3.2455
4	4h	26	-31.78%	0.3356	-3.4163	-29.25%	0.3102	-3.3508	-23.24%	0.2661	-3.2455
5	1d	27	-30.85%	0.3208	-3.2586	-27.89%	0.2937	-3.1974	-23.24%	0.2661	-3.2455

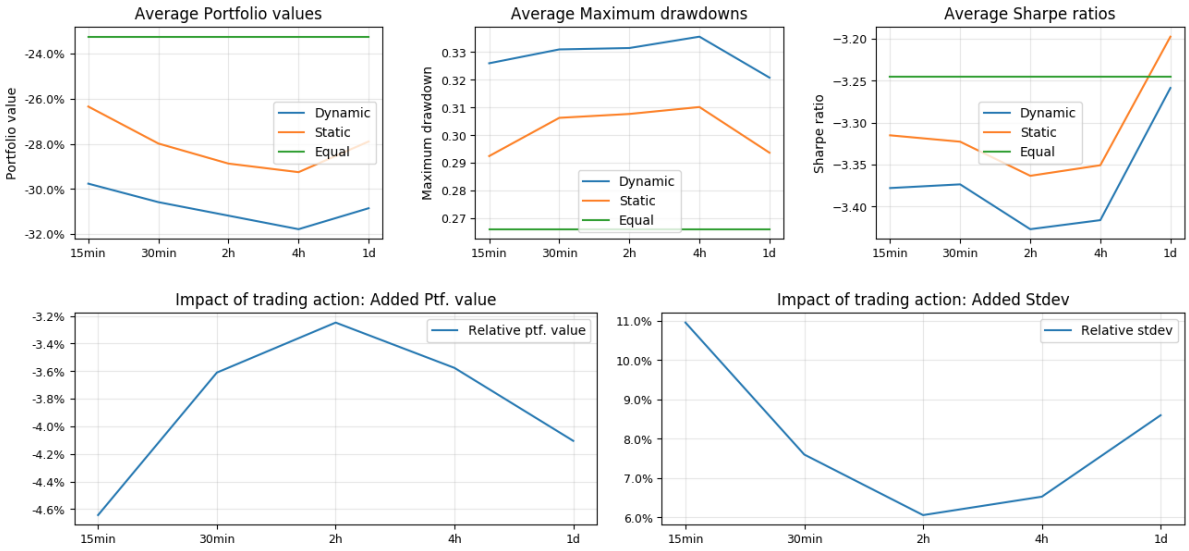


Figure 8: Aggregated report example

5.5.2. Simulation summary

The simulation summary displays the performance of a single simulation. We generated over a thousand of these reports and picked some of the more interesting for this final report. In chapter 5.1 we already showed an example of a simulation summary report. The simulation summary report consists of key statistics of a simulation run. In addition, it has charts that visualize relevant price action and weight evolution. Please refer to section 5.1 and "Figure 6: Simulation summary example" for an example.

5.5.3. Simulation stability report

Since deep learning models are highly stochastic, we were concerned about the stability of our results. Therefore we created a simulation stability report that compares the results of the identical simulations. These reports collect all the simulations and plots histograms of some key metrics. Furthermore, we calculated the standard deviation of the metrics and present them in the tables on the top. Turns out that our simulations were most of the time

remarkably stable. The stability reports can be found from the appendices. Below is an example report (Figure 9).

[Simulation summary] Rock bottom 2h

Dynamic agent	Average	Stdev
Ptf. value	1.1245	0.0015
Sharpe ratio	3.3298	0.016
Sharpe ratio (ann)	8.9966	0.0433
MDD	0.1197	0.0006
Average of weights	0.0873	0.0002
Stdev of weights	0.0016	0.0
Cash weight (BTC)	0.0392	0.0027

Static agent	Average	Stdev
Ptf. value	1.0923	0.0004
Sharpe ratio	2.3501	0.0099
Sharpe ratio (ann)	6.3496	0.0267
MDD	0.2006	0.0006
Average of weights	0.0873	0.0002
Stdev of weights	0.0016	0.0
Cash weight (BTC)	0.0392	0.0027

Equal weighted	Average
Ptf. value	1.0903
Sharpe ratio	2.4133
Sharpe ratio (ann)	6.7327
MDD	0.1923

Parameter	Value
No. of simulations	27
No. of assets	11
Start date	2018-11-10
End date	2018-12-31
Trading period	2h

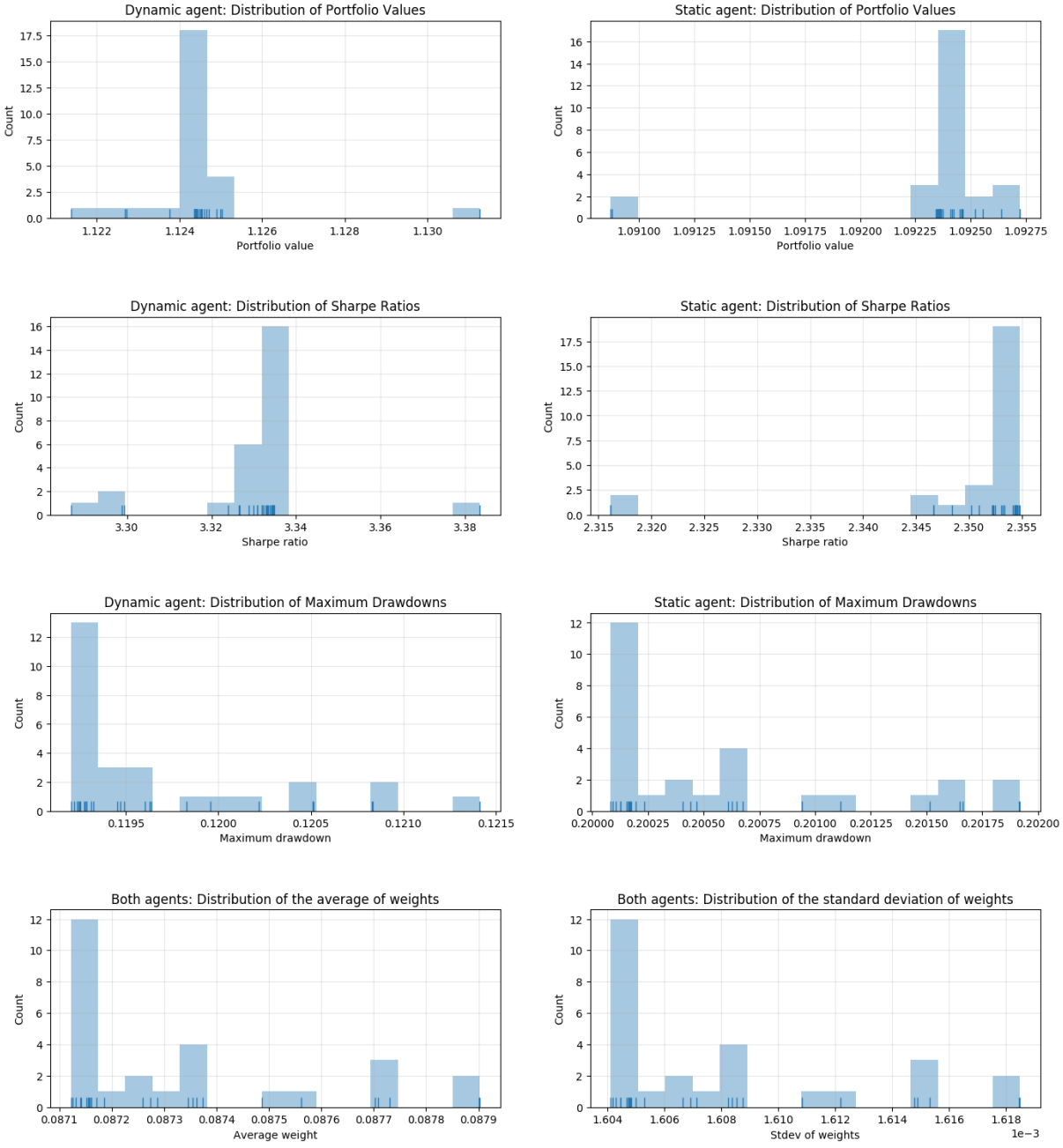


Figure 9: Simulation stability report example

5.6. Backtests

5.6.1. Calm before the storm

2016-09-07 to 2016-10-28

The first backtest is the least dramatic of the backtests and was mainly chosen, because it was the first backtest period of Jiang et al. (2017). This is still from the time where the cryptocurrency market was in its infancy. Bitcoin did not yet have any credible competition and held over 80% of the whole market capitalization. Back then the authors would not have recommended to diversify into any other cryptocurrency apart from Bitcoin.

During this 50-day back test period the market starts to show some serious confidence again in Bitcoin and the total market capitalization grows from 11.6 billion USD to 13.3 billion USD. Bitcoin leads the show by increasing its dominance from 80.08% to 82.190%; the second highest growth of all the backtests. From this growth we can already see that the Sharpe ratio will be negative for this period.

Analysis

"Calm before the storm" was the least profitable of all the backtests, since all agents lost at least 23% of the portfolio value. The unprofitability was expected, due to the large increase of Bitcoin dominance. This was magnified by poor asset selection: the portfolios held STEEM and XMR during the period that performed particularly poor. Moreover, every asset in the portfolio had negative returns against Bitcoin during the period.

We were unable to replicate Jiang et al.'s portfolio value as they claimed to achieve 4- to 29-fold returns during this period. This is curious, as we are confident that we used the exactly same assets and time range. We find it incredible that they could have achieved so impressive returns while all the assets managed lose value. We had also significantly higher MDDs (0.33 vs 0.22).

The equal weighted strategy outperformed both of the agents during this period. This is due to the agent's tendency to invest less on the cash asset (Bitcoin). Since Bitcoin's dominance grew in this period, the equal weighted suffered less losses since it invests a larger amount in cash. This will be a recurring theme in all the backtests.

The dynamic agent did particularly poor on this period. We can see from the impact of trading action charts that the trading action destroyed value and added volatility (Figure 10). Since the dynamic agent pursues to stabilize weights, it constantly invests more in STEEM and XMR as they plummet to new lows. Contrarily the static agent does not adjust its weights when the relative weights of STEEM and XMR fall.

From the simulation report (Figure 11) we can clearly see that dynamic agent loses more and more value towards the end of the backtest period as STEEM continues to collapse. This is awarded with a very high maximum drawdown. This backtest is a prime example of when the auto-balancing strategy does not work since prices do not mean revert. From the stability report (see appendices) we can see that the results were quite stable. The standard deviation of weights (not including cash) is in the fourth decimal so all the differences are due to different cash weights.

We can also note that the 2 h trading action destroyed least value as it both destroyed least value and added the least amount of volatility (Figure 10).

Calm before the storm			Dynamic agent			Static agent			Equal weighted		
#	Period	Simulations	Ptf. value	MDD	Sharpe	Ptf. value	MDD	Sharpe	Ptf. value	MDD	Sharpe
1	15min	21	-29.76%	0.326	-3.378	-26.34%	0.2925	-3.3151	-23.24%	0.2661	-3.2455
2	30min	19	-30.58%	0.331	-3.3736	-27.98%	0.3063	-3.3227	-23.24%	0.2661	-3.2455
3	2h	33	-31.18%	0.3315	-3.4271	-28.87%	0.3077	-3.3634	-23.24%	0.2661	-3.2455
4	4h	26	-31.78%	0.3356	-3.4163	-29.25%	0.3102	-3.3508	-23.24%	0.2661	-3.2455
5	1d	27	-30.85%	0.3208	-3.2586	-27.89%	0.2937	-3.1974	-23.24%	0.2661	-3.2455

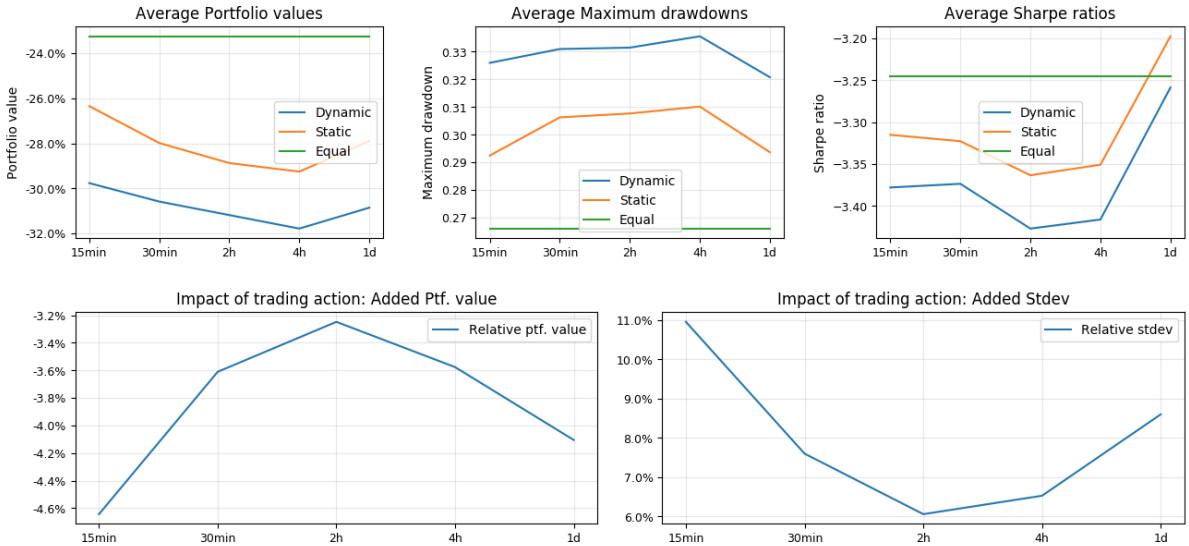


Figure 10: Calm before the storm - Aggregated report

Calm before the storm 15min

Strategy	Ptf value	Sharpe	Sharpe (ann.)	MDD
Dynamic agent	0.7025	-3.3782	-9.1274	0.3259
Static agent	0.7369	-3.3157	-8.9586	0.2921
Equal weighted	0.7676	-3.2455	-8.7687	0.2661

Dataset	Start date	End date	Days	Steps
Train period	2016-01-03	2016-09-06	247	23816
Test period	2016-09-07	2016-10-28	50	4884

Parameter	Value
No. batches	20
No. episodes	3
Batch size	50
Trading fee	0.2%
Trading period	15min
Window length	70
Conv Filters, 1	5
Conv Filters, 2	50
Kernel size	(1, 3)
ε greedy thld	0.8

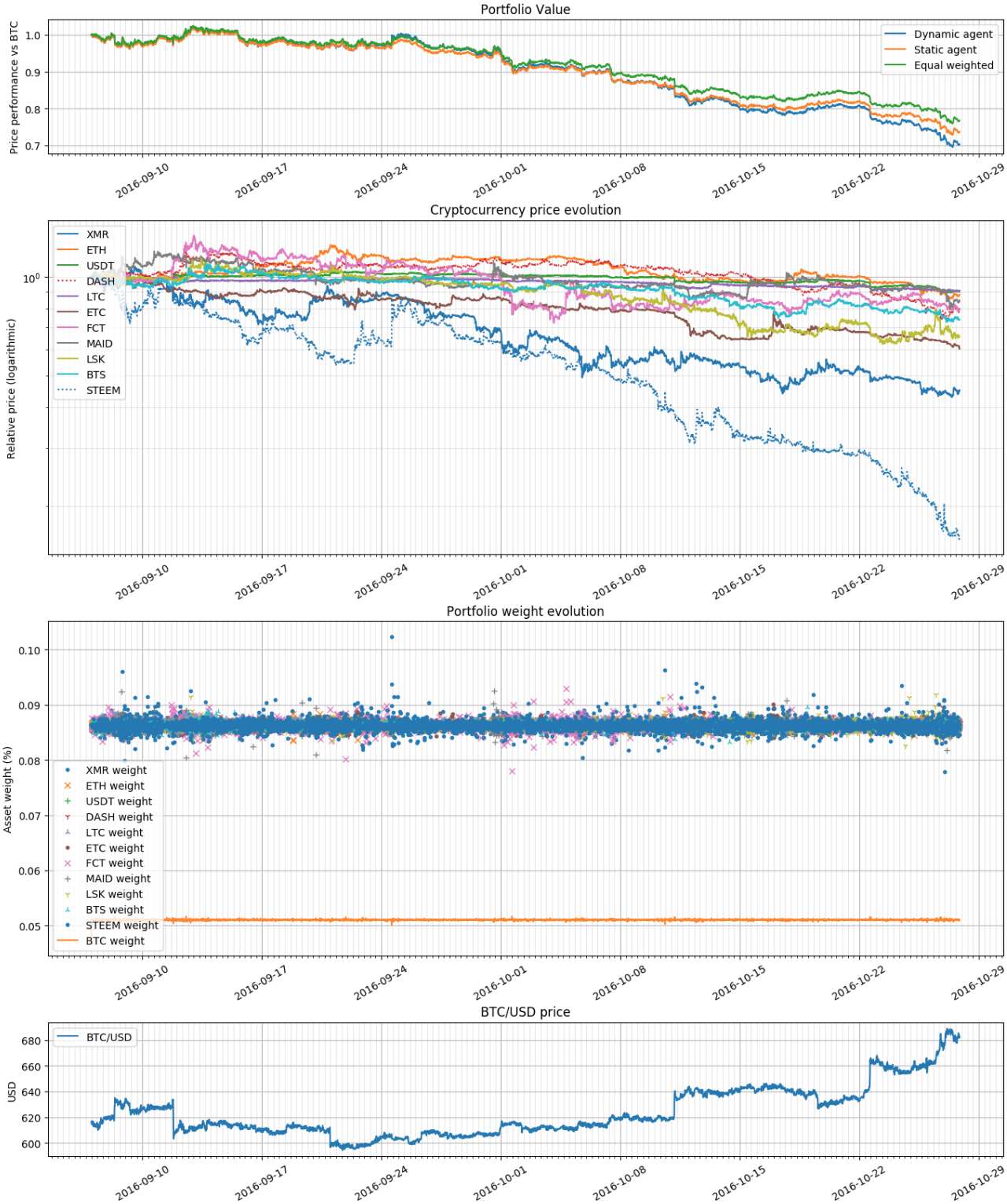


Figure 11: Calm before the storm - Simulation summary (15 min example)

5.6.2. Awakening

2016-12-08 to 2017-01-28

During this period things start to get quite interesting. Bitcoin's price grows over 55.5% to reach its all-time levels. The cryptocurrency markets are warming up for a crazy year. The show is led by Bitcoin; it has the highest dominance of all the backtest periods with a high value of 88.11%. However, we can expect a slightly positive Sharpe ratio since Bitcoin loses some dominance during this 50 day period. The total market capitalization of cryptocurrencies reaches a new height of 21.9 billion USD. This was also the second backtest of Jiang et al. (2017).

Analysis

During the "Awakening", the relative prices against stayed pretty stable and there was least price oscillation from all periods. Since Bitcoin lost some of its dominance, all portfolios experienced growth. Greatest losers of the period where STEEM and ZEC.

While the peak and through differences between asset prices remained quite low as indicated by the low maximum drawdown (Figure 12), there was still a lot of mean reverting behavior as can be seen from the simulation report (Figure 13). Especially around 2017-01-10 to 2017-01-24 the dynamic agent was able to benefit from this from time intervals 15 min to 4h. However, the trading action did add quite a lot of volatility as stated by the "Impact of trading action charts" (Figure 12). Further, the 1 day trading action seemed to destroy value.

Another interesting observation is that static agents portfolio value decreased from period 15 min to period 1 day. This is due to the fact that for an unknown reason the deep learning agent gave on average a larger weight on Bitcoin for the longer trade periods.

It is not straightforward which agent performed the best during this period. On average the agents' weight initialization worked better than the equal weight for periods 15 min to 30 min but after that the equal weighted worked better (Figure 12). The dynamic agent was able to create value especially on the shorter periods but with a higher variance.

While at Jiang et al.'s study this was the least profitable backtest, they were still able to achieve 1.5 to 8x returns with MDD around 0.25. Again, we could not replicate their results.

Our MDDs were on the same ballpark but lower. Again, it feels amazing that they could have achieved so high returns with this price action.

From the stability report we can note that the results were again quite stable. (See appendices)

From the "Impact of trading action" charts we can observe that 30 min rebalancing worked best. It increased most return, but also had a high volatility. (Figure 12)

Awakening			Dynamic agent			Static agent			Equal weighted		
#	Period	Simulations	Ptf. value	MDD	Sharpe	Ptf. value	MDD	Sharpe	Ptf. value	MDD	Sharpe
6	15min	19	11.32%	0.1623	1.356	10.16%	0.1907	1.4036	8.61%	0.1769	1.3021
7	30min	19	10.33%	0.161	1.2831	9.09%	0.1905	1.3005	8.61%	0.1769	1.3021
8	2h	34	9.21%	0.158	1.1896	8.35%	0.1838	1.2238	8.61%	0.1769	1.3021
9	4h	34	8.59%	0.1548	1.1515	8.14%	0.1756	1.2213	8.61%	0.1769	1.3021
10	1d	33	6.95%	0.1387	0.9645	7.44%	0.1553	1.1198	8.61%	0.1769	1.3021

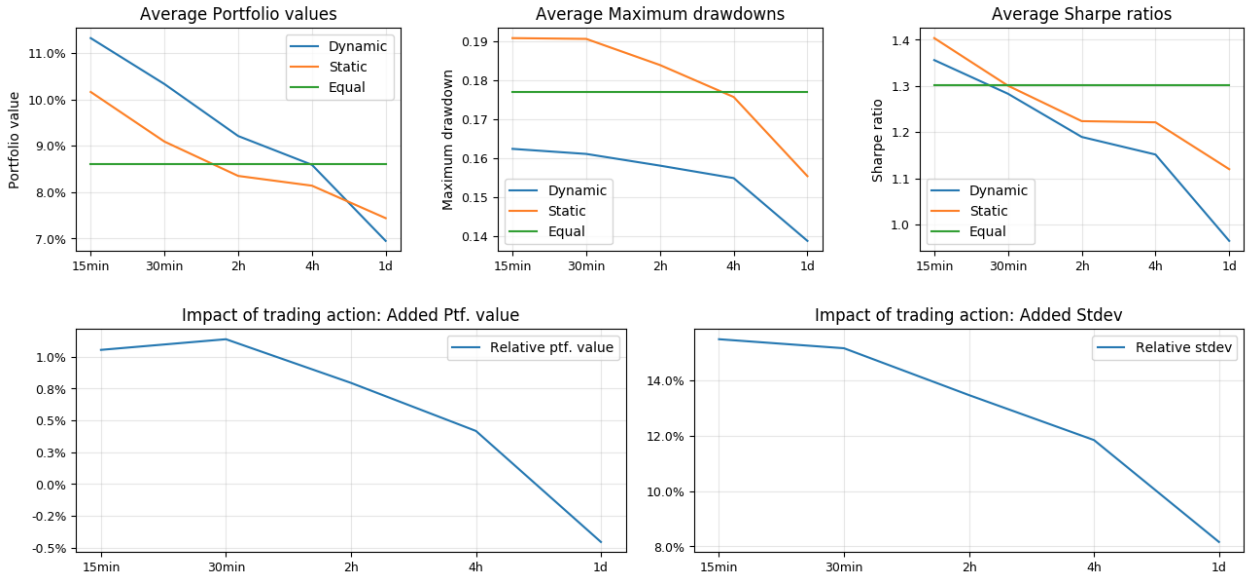


Figure 12: Awakening - Aggregated report

.

Awakening 2h

Strategy	Ptf value	Sharpe	Sharpe (ann.)	MDD
Dynamic agent	1.0912	1.209	3.2665	0.1529
Static agent	1.0824	1.2424	3.3567	0.1777
Equal weighted	1.0861	1.3021	3.5181	0.1769

Dataset	Start date	End date	Days	Steps
Train period	2015-06-09	2016-12-07	547	6578
Test period	2016-12-08	2017-01-28	50	611

Parameter	Value
No. batches	20
No. episodes	3
Batch size	50
Trading fee	0.2%
Trading period	2h
Window length	70
Conv Filters, 1	5
Conv Filters, 2	50
Kernel size	(1, 3)
ε greedy thld	0.8

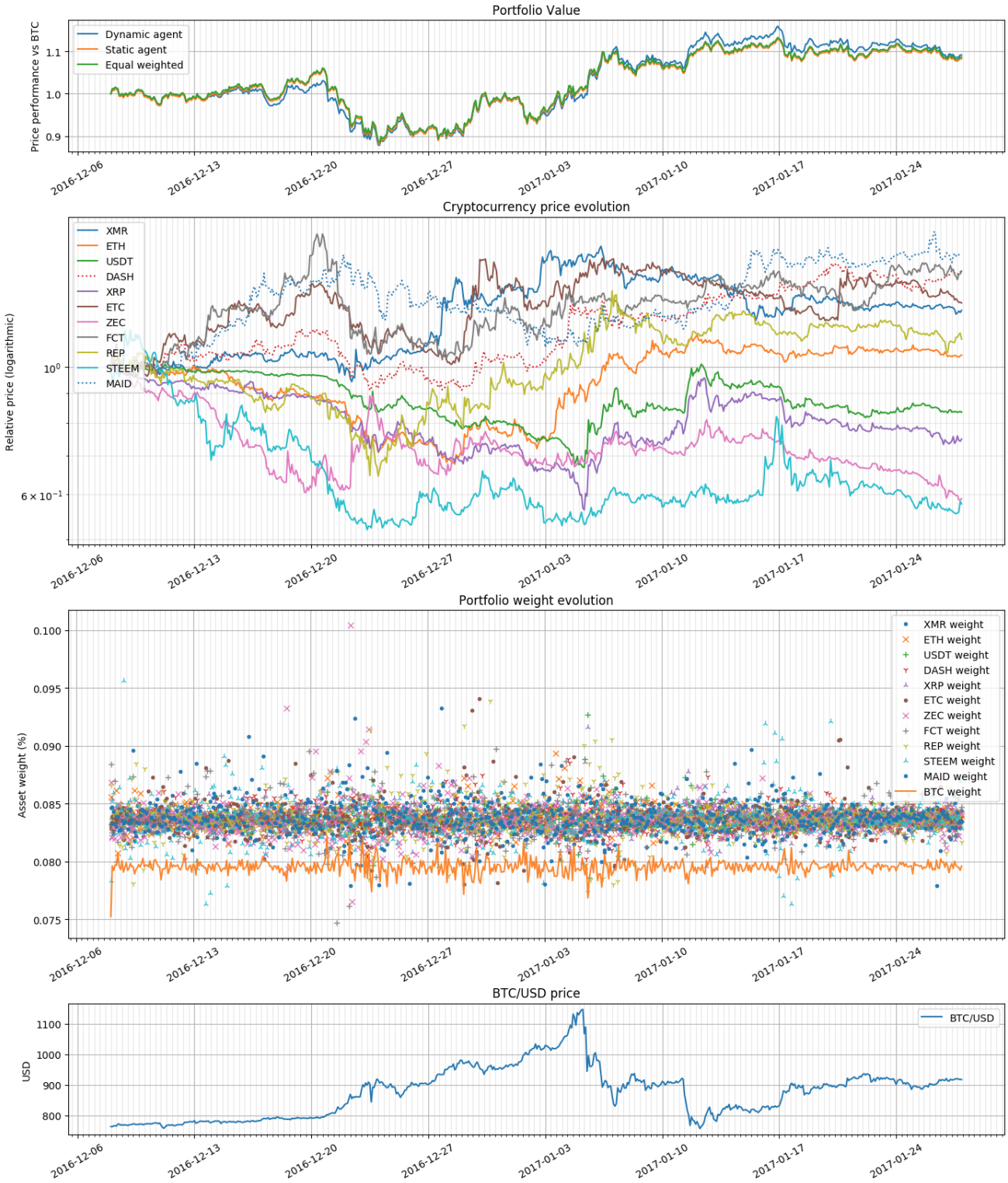


Figure 13: Awakening - Simulation summary (2 hour example)

5.6.3. Ripple bull run

2017-03-07 to 2017-04-27

This 50 day period is highly remarkable in the history of cryptocurrencies. It seems probable that this was the last time that Bitcoin had an over 80% dominance. We can expect a high Sharpe ratio as Bitcoin loses almost 22% of its dominance. However this loss is not due to the plummeting of Bitcoin price. (Bitcoin's price actually slightly increases during the period.) The loss is due to two competitors, Ethereum (ETH) and Ripple (XRP) gaining some major valuation increase. The total market capitalization of cryptocurrencies grows from 24.4 billion USD to 31.6 billion USD. This was also the third backtest of Jiang et al. (2017). They probably specifically chose it because it is so special.

Analysis

During the "Ripple bull run", the portfolio values grew the most of all backtests as the portfolios benefited from Bitcoins loss of dominance. However, there was very high volatility which can be witnessed from the high maximum drawdowns. This also lowered the Sharpe ratios.

At Jiang et al.'s study this was the most profitable backtest and they claim to have achieved 4 to 47 fold returns. Our maximum return of 1.7-fold is quite modest compared to that. (Figure 14) However, the MDD of our dynamic agent was quite near to theirs (ours: 0.3538, theirs: 0.406). Still it is incredible if they could 47-fold their returns when the best asset in their portfolio only 10-folded its value.

This backtest is a prime example of how the dynamic agent excels when prices are mean reverting. We showed how the dynamic agent excels in mean reverting conditions in section 5.1. Therefore we want to concentrate here on the 4h example and have a look why the dynamic and static agent performed so badly there. The problem is due to unstable portfolio weight initialization and it occurred in the 4 h and 1 day periods.

Ripple bull run			Dynamic agent			Static agent			Equal weighted		
#	Period	Simulations	Ptf. value	MDD	Sharpe	Ptf. value	MDD	Sharpe	Ptf. value	MDD	Sharpe
11	15min	20	169.42%	0.3696	3.1172	151.73%	0.8716	3.1088	144.69%	0.6559	3.0794
12	30min	23	163.69%	0.3538	3.1168	149.88%	0.8572	3.1025	144.69%	0.6559	3.0794
13	2h	29	154.34%	0.3518	3.099	150.35%	0.8586	3.0678	144.69%	0.6559	3.0794
14	4h	37	142.97%	0.2905	3.1081	145.07%	0.6594	3.0717	144.69%	0.6559	3.0794
15	1d	24	168.6%	0.2961	2.9863	150.38%	0.6991	2.964	144.69%	0.6559	3.0794

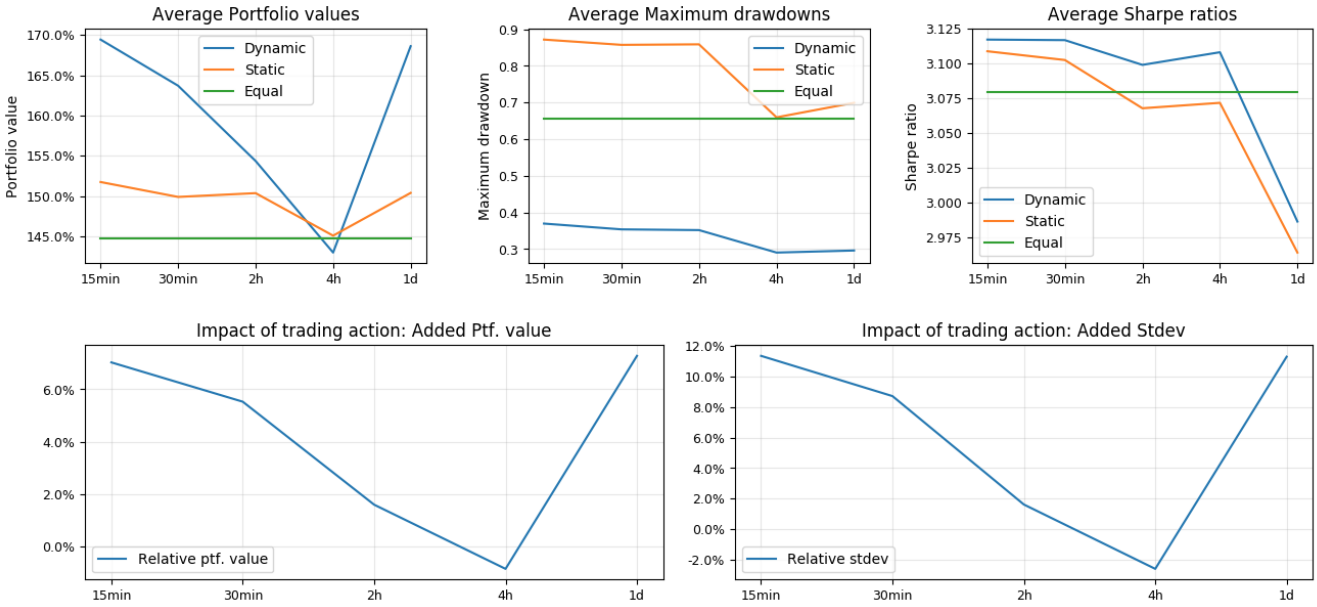


Figure 14: Ripple bull run - Aggregated report

We present a simulation summary from the 4h period (Figure 15) where the initialization was not even. Here we can witness the deep learning model trying to do something different to outperform the market. In this case it does a great job. It gives high weights to the most profitable assets of the period. Notice the weights in the Portfolio weight evolution chart that are not even. This feature is amplified in the dynamic agent, as it wants to go back to its high Ripple target weight and aggressively buys it as the price soars. However, there were a lot of inverse examples that lead to the low average return for the 4h period.

Due to instability issues, it is difficult to determine which rebalancing period worked the best. All rebalancing action seemed to be quite profitable. However, we cannot draw conclusions regarding the best period.

Ripple bull run 4h

Strategy	Ptf value	Sharpe	Sharpe (ann.)	MDD
Dynamic agent	2.9623	3.1395	8.4826	0.3239
Static agent	2.6929	3.0752	8.3086	0.7092
Equal weighted	2.4469	3.0794	8.3201	0.6559

Dataset	Start date	End date	Days	Steps
Train period	2015-09-06	2017-03-06	547	3289
Test period	2017-03-07	2017-04-27	50	305

Parameter	Value
No. batches	20
No. episodes	3
Batch size	50
Trading fee	0.2%
Trading period	4h
Window length	70
Conv Filters, 1	5
Conv Filters, 2	50
Kernel size	(1, 3)
ε greedy thld	0.8

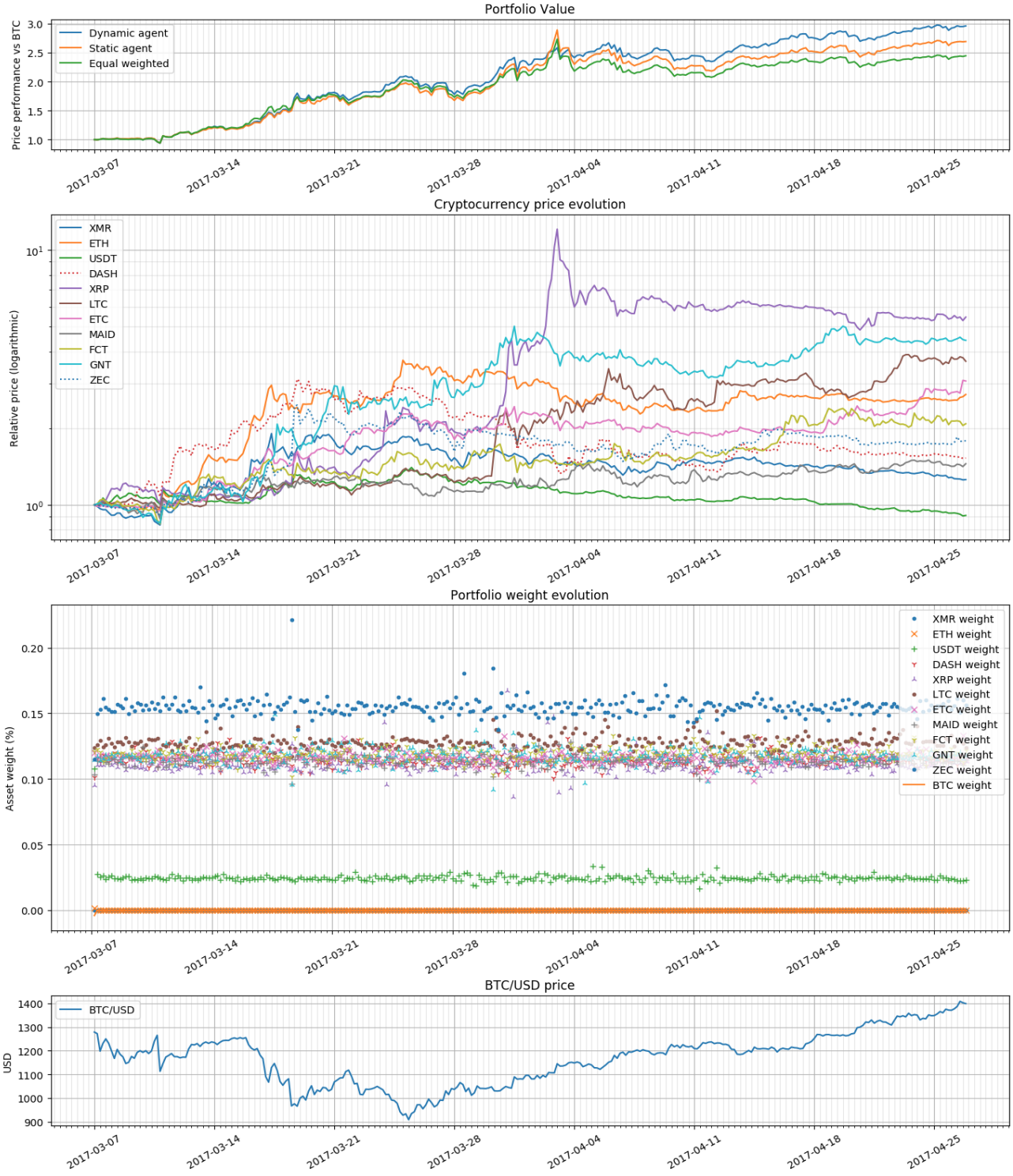


Figure 15: Ripple bull run - Simulation summary (4 hour example)

5.6.4. Ethereum valley

2017-05-28 to 2017-07-18

Similar to the last backtest, this 50 day period is seminal in the history of cryptocurrencies. As of April 2019, this was the only period when Bitcoin's position as the most valuable cryptocurrency was seriously challenged. This was due to the bull run of Ethereum which had already begun on the start of the year. The margin between Bitcoin and Ethereum was at its lowest below 6 percentage points. Bitcoin starts this period with a market capitalization of 48.7%, goes to a low of 37.1% and finishes with 49.4%. Ethereum performs a similar pattern but reversed. (Hence the name "Ethereum valley".) During this period the total market capitalization of cryptocurrencies reaches its all-time high of 116 billion USD.

Analysis

The "Ethereum valley" backtest period was deliberately chosen to demonstrate the environment were the dynamic agent excels. It included the greatest price oscillation of all periods. While Bitcoin dominance reached its all-time low during this period, at the end of it reached its starting value. Therefore it has high volatility, high MDDs and low returns. These combine into low Sharpe ratios (Figure 16).

This type of price action is optimal for the dynamic agent and it strictly outperformed the other strategies. In fact, it was the only strategy that had a positive portfolio value and Sharpe ratio. The value was purely created by trading action as can be verified from the "Impact of trading action" charts (Figure 16). The period starts by Ripple (XRP) price increase during 2017-06-02 which maintains for a week. However after 2017-06-20 many portfolio assets start to oscillate a lot and after that the dynamic portfolio value stays on top of its benchmarks (Figure 17). Most of the final value gain occurs in the last week due to the upward and downward price action.

While this is a prime example of our dynamic strategy at its best, we do not claim to be able predict when these kind of periods might occur.

From the "Impact of trading action" (Figure 16) charts we can notice that 30 min rebalancing created most value. However, 2 h rebalancing has the lowest volatility. Therefore we conclude that both of these rebalancing periods had their strengths and do not choose one or the other.

Ethereum valley			Dynamic agent			Static agent			Equal weighted		
#	Period	Simulations	Ptf. value	MDD	Sharpe	Ptf. value	MDD	Sharpe	Ptf. value	MDD	Sharpe
16	15min	32	4.08%	0.4782	0.4542	-1.21%	0.4645	-0.1251	-0.82%	0.4434	-0.0898
17	30min	35	3.21%	0.4742	0.3611	-2.1%	0.4633	-0.2182	-0.82%	0.4434	-0.0898
18	2h	36	1.52%	0.4541	0.1723	-2.93%	0.4465	-0.3055	-0.82%	0.4434	-0.0898
19	4h	29	0.88%	0.4515	0.0997	-3.03%	0.4416	-0.317	-0.82%	0.4434	-0.0898
20	1d	24	1.37%	0.4012	0.1441	-1.52%	0.3898	-0.1545	-0.82%	0.4434	-0.0898

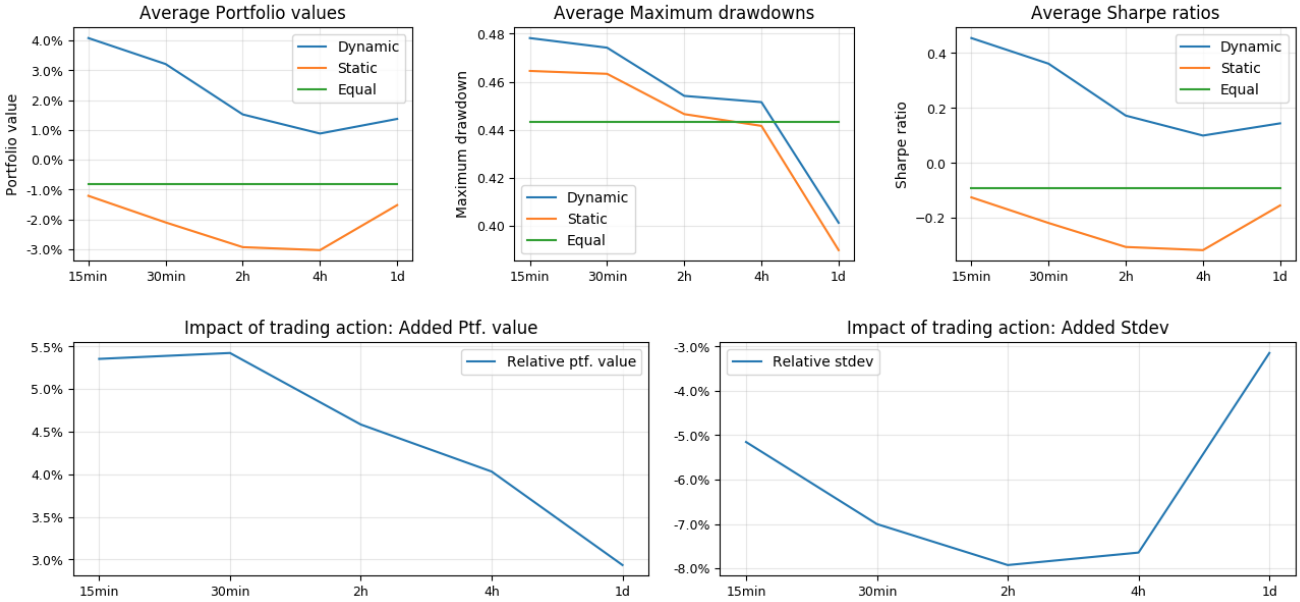


Figure 16: Ethereum valley - Aggregated report

Ethereum valley 30min

Strategy	Ptf value	Sharpe	Sharpe (ann.)	MDD
Dynamic agent	1.0379	0.4424	1.1952	0.4565
Static agent	0.9865	-0.1467	-0.3965	0.4457
Equal weighted	0.9918	-0.0898	-0.2427	0.4434

Dataset	Start date	End date	Days	Steps
Train period	2015-11-27	2017-05-27	547	26308
Test period	2017-05-28	2017-07-18	50	2442

Parameter	Value
No. batches	20
No. episodes	3
Batch size	50
Trading fee	0.2%
Trading period	30min
Window length	70
Conv Filters, 1	5
Conv Filters, 2	50
Kernel size	(1, 3)
ε greedy thld	0.8

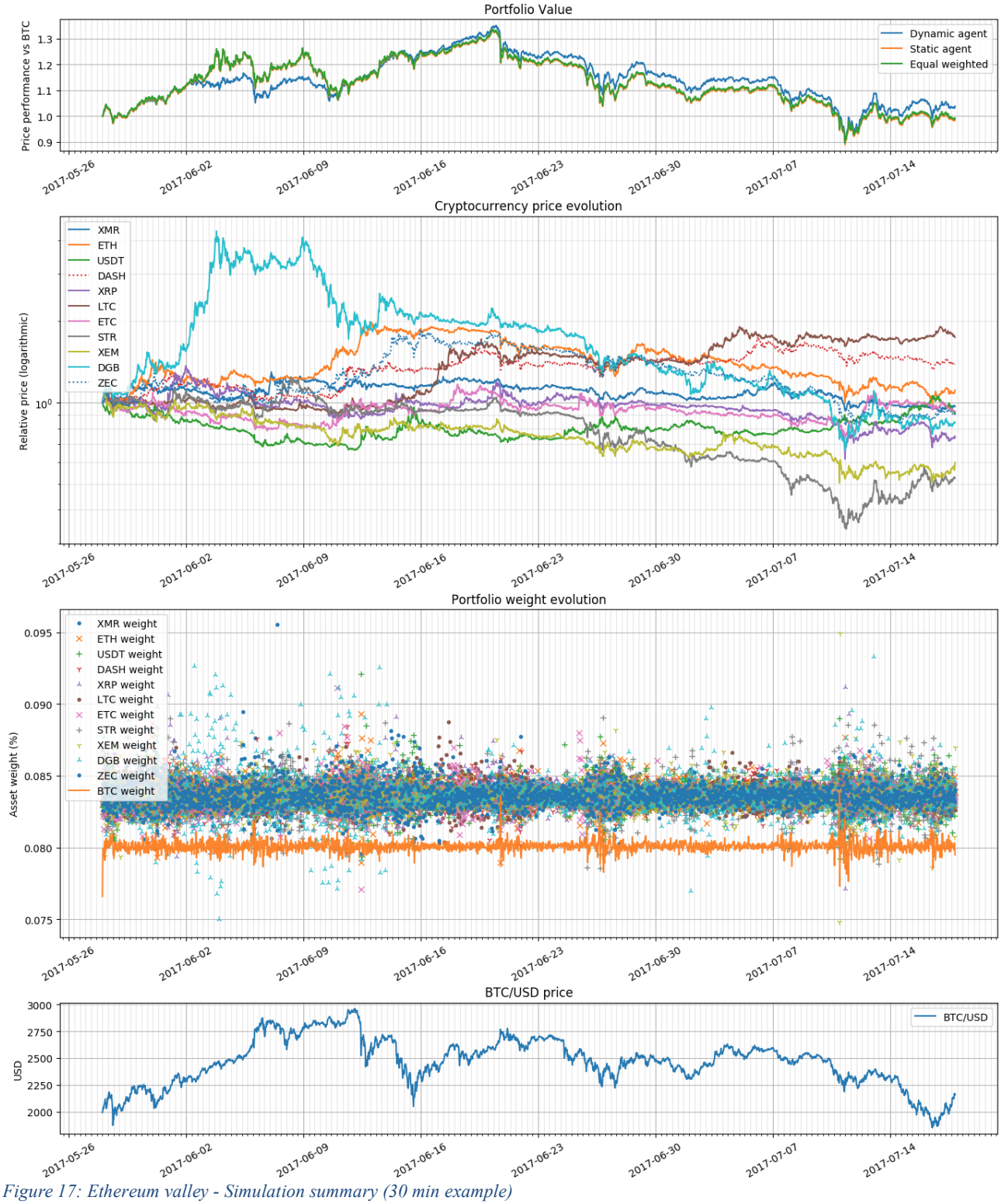


Figure 17: Ethereum valley - Simulation summary (30 min example)

5.6.5. All-time high

2017-11-23 to 2018-01-13

This was the period when exuberance reached its peak. As of April 2019, this was the period when Bitcoin reached its all-time high of nearly 20.000 USD while starting from a value of 8077.95 USD. However, other cryptocurrencies also displayed very remarkable price action as Bitcoin's dominance also reached its all-time low of 32.51%. The loss in dominance was due to price increases of various coins but was once again spearheaded by Ethereum and Ripple. Total market capitalization of cryptocurrencies reaches its all-time high of 831 billion USD.

Analysis

During the "All-time high", the portfolios experienced their second highest growth after "Ripple bull run". There was also a lot of price oscillation, but unlike in "Ripple bull run", the prices do not mean revert towards the end of the period. The MDDs of this period as well as the Sharpe ratios are also very high. (Figure 18)

This backtest period is a good example on how the dynamic agent works as a conservative strategy in highly volatile market conditions. While all the strategies benefit greatly from Bitcoin's dominance reaching its all-time low, the dynamic agent misses some of the gains created by Ethereum and Ripple as they stay at high price levels towards the end of the period. (Figure 19)

The dynamic agent has a lower return but also a lower volatility at all trading periods. This is shown by the Added Stdev chart (Figure 18), where we can see that the strategy reduced the standard deviation by an maximum of 30% at its maximum. This is awarded as an impressive Sharpe ratio which is very similar in magnitude to the equal weighted and static agent.

It is really up to the risk appetite of the investor to determine which strategy performed the best during this crazy period. From the "Impact of trading action" charts we can clearly conclude that 30 min trading action created most value. (Figure 18)

All-time high			Dynamic agent			Static agent			Equal weighted		
#	Period	Simulations	Ptf. value	MDD	Sharpe	Ptf. value	MDD	Sharpe	Ptf. value	MDD	Sharpe
21	15min	18	91.3%	0.4596	2.2927	134.14%	0.7591	2.6293	128.24%	0.6324	2.6234
22	30min	10	103.58%	0.4299	2.656	133.1%	0.7611	2.621	128.24%	0.6324	2.6234
23	2h	25	95.65%	0.4301	2.6202	132.21%	0.6549	2.6146	128.24%	0.6324	2.6234
24	4h	25	90.08%	0.3967	2.5786	129.22%	0.624	2.5952	128.24%	0.6324	2.6234
25	1d	24	90.87%	0.4015	2.4836	122.11%	0.5634	2.4527	128.24%	0.6324	2.6234

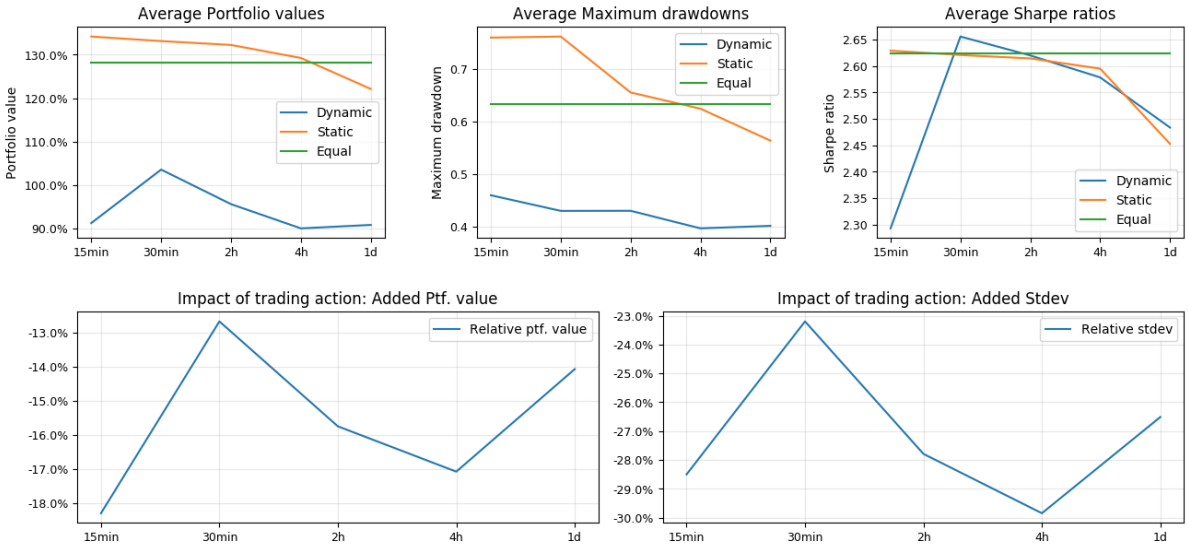


Figure 18: All-time high - Aggregated report

All-time high 2h

Strategy	Ptf value	Sharpe	Sharpe (ann.)	MDD
Dynamic agent	1.9544	2.62	7.0789	0.4293
Static agent	2.3195	2.6147	7.0646	0.6536
Equal weighted	2.2824	2.6234	7.0881	0.6324

Dataset	Start date	End date	Days	Steps
Train period	2016-05-24	2017-11-22	547	6578
Test period	2017-11-23	2018-01-13	50	611

Parameter	Value
No. batches	20
No. episodes	3
Batch size	50
Trading fee	0.2%
Trading period	2h
Window length	70
Conv Filters, 1	5
Conv Filters, 2	50
Kernel size	(1, 3)
ε greedy thld	0.8

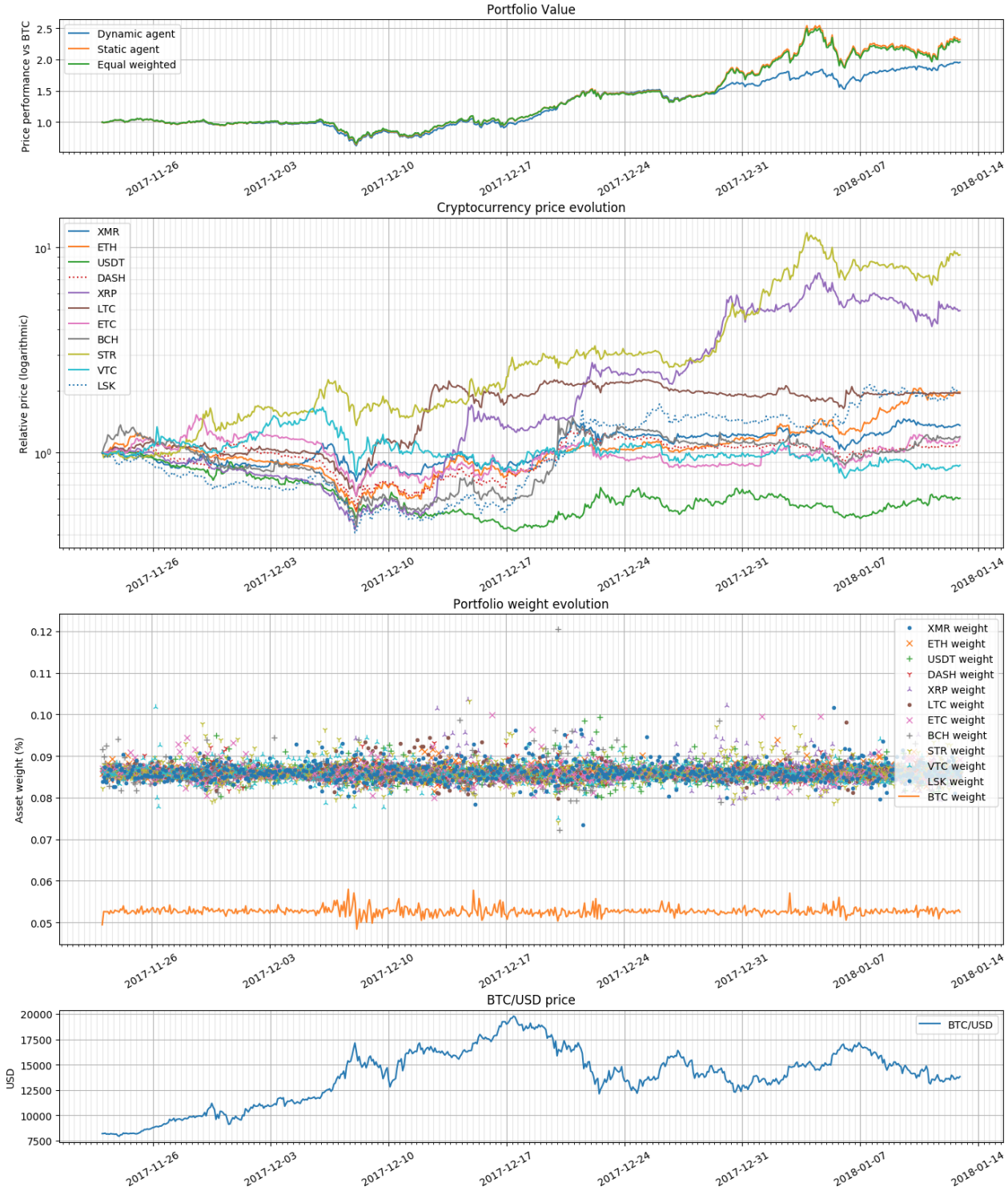


Figure 19: All-time high - Simulation summary (2 hour example)

5.6.6. Rock bottom

2018-11-10 to 2018-12-31

Towards the end of 2018 Bitcoin lost almost 85% of its value by reaching a low of 3199.01 USD. Only during this 50 day period, Bitcoin loses almost 40% of its value starting from 6411.76 USD and ending with 3865.95 USD. All other cryptocurrencies lose a similar amount of market capitalization: Bitcoin's dominance decreases only slightly. The optimism of 2017 has vanished from the cryptocurrency market as Bitcoin's dominance has grown back to 55%. Total market capitalization starts at 210 billion USD and ends with 130 billion USD.

Analysis

During "Rock bottom" the total amount of volatility in the portfolio is at its lowest. When combined with slight decrease of Bitcoin dominance, we get a good return and very impressive Sharpe ratios. The MDDs are lowest of all backtests. (Figure 20)

This backtest period is an great example of how the dynamic agent does not need a lot of volatility to excel as long as there are mean reverting conditions. During this period, the price of Bitcoin Cash SV (BCHSV) varied significantly as indicated with a yellow line in the simulation summary. Consequently the dynamic agent strictly outperformed the static and equal weighted strategies in all indicators. (Figure 21)

As can be seen from the Impact of trading action charts (Figure 20), the trading action created value at all intervals and peaked at 2h and 1d. It was also able to reduce the volatility by at least 4%. From the "Impact of trading action" charts we can conclude that 2h or the 1 day trading action created most value. (Figure 20)

Rock bottom			Dynamic agent			Static agent			Equal weighted		
#	Period	Simulations	Ptf. value	MDD	Sharpe	Ptf. value	MDD	Sharpe	Ptf. value	MDD	Sharpe
26	15min	36	11.05%	0.1156	3.0593	10.47%	0.2167	2.4907	9.4%	0.1916	2.4698
27	30min	26	10.63%	0.1179	2.9665	9.13%	0.2138	2.2679	9.4%	0.1916	2.4698
28	2h	25	12.17%	0.1211	3.2144	9.19%	0.2002	2.3314	9.4%	0.1916	2.4698
29	4h	28	10.41%	0.1248	3.0405	9.56%	0.1771	2.4209	9.4%	0.1916	2.4698
30	1d	20	12.39%	0.1018	3.3833	9.38%	0.1563	2.4586	9.4%	0.1916	2.4698

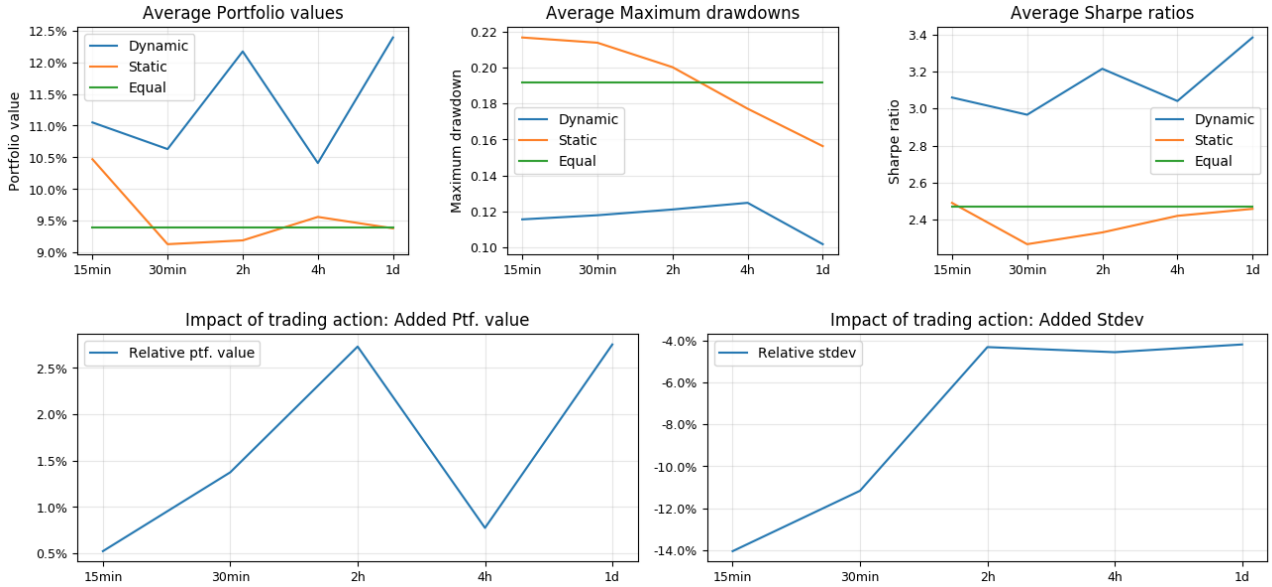


Figure 20: Rock bottom - Aggregated report

Rock bottom 2h

Strategy	Ptf value	Sharpe	Sharpe (ann.)	MDD
Dynamic agent	1.1245	3.3319	9.0024	0.1196
Static agent	1.0925	2.3523	6.3555	0.2006
Equal weighted	1.0945	2.4919	6.7327	0.1918

Dataset	Start date	End date	Days	Steps
Train period	2017-05-11	2018-11-09	547	6580
Test period	2018-11-10	2018-12-31	50	609

Parameter	Value
No. batches	20
No. episodes	3
Batch size	50
Trading fee	0.2%
Trading period	2h
Window length	70
Conv Filters, 1	5
Conv Filters, 2	50
Kernel size	(1, 3)
ε greedy thld	0.8

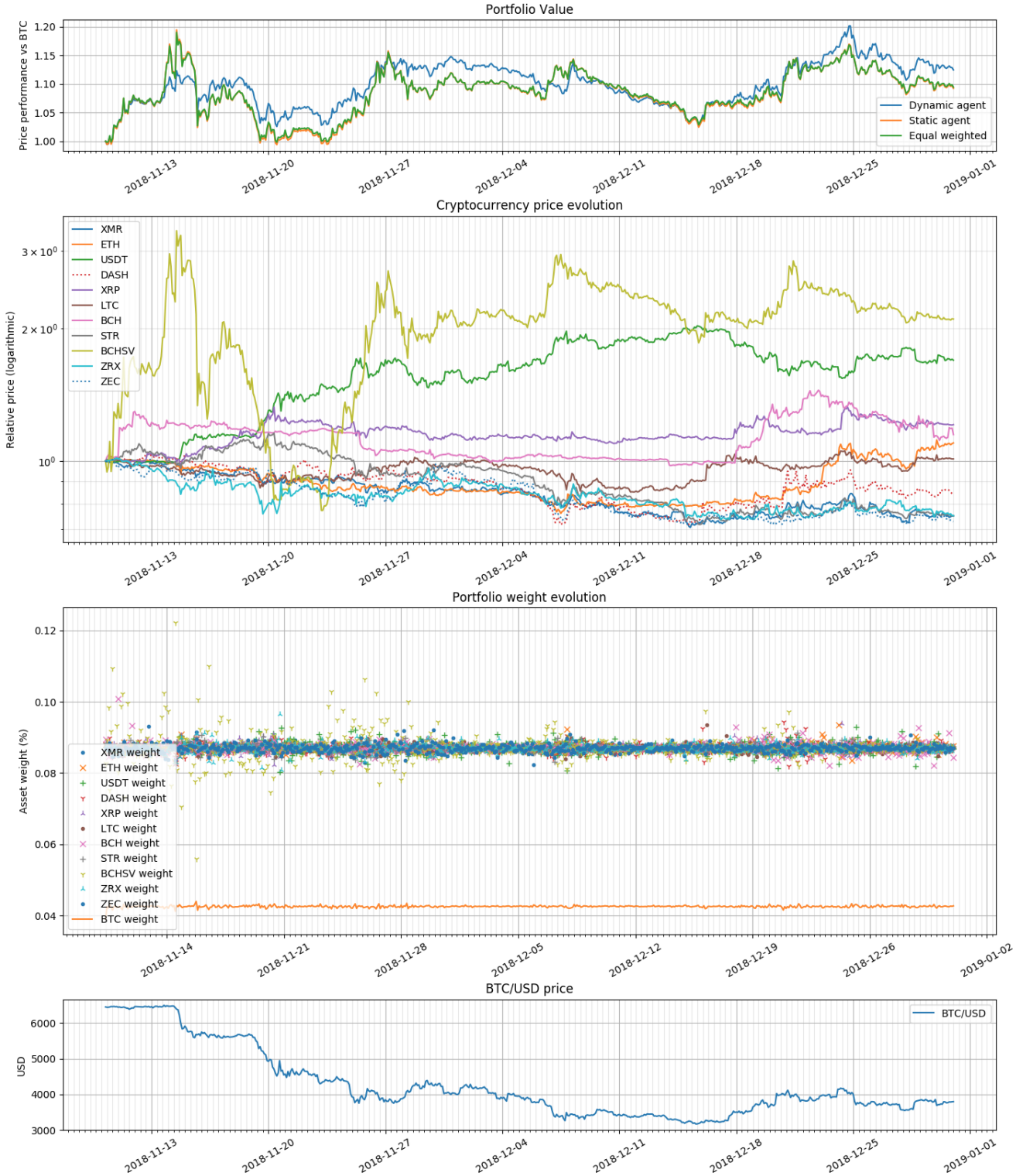


Figure 21: Rock bottom - Simulation summary (2 hour example)

5.6.7. Recent

2019-03-06 to 2019-04-26

This period was mainly chosen, because it is the most fresh data available at the time of writing this thesis. Some careful optimism has returned to the cryptocurrency space. Bitcoin's price has increased from a start of 3913.23 USD to an end value of 5281.63 USD. While many major cryptocurrencies have also increased in valuation, this period of growth has been clearly led by Bitcoin. This is the backtest period with the largest overall Bitcoin dominance growth. Therefore severely negative Sharpe ratios are expected. Total market capitalization of cryptocurrencies grows from 132 billion USD to 168 billion USD.

Analysis

During "Recent" backtest period, the portfolios experience the second sharpest loss after "Calm before the storm" due to the growth of Bitcoin dominance. The Sharpe ratios are also very low. There is not a lot of overall volatility in the market and the MDDs are quite low. (Figure 22)

During this backtest period the equal weighted strategy performed clearly the best. This is mainly due to the dynamic and static agents' tendency to hold less weight on the cash asset (Bitcoin) than on the other assets. Therefore the equal weighted strategy took least damage from Bitcoin's dominance growth. (Figure 23)

Furthermore, the static agent outperforms the dynamic agent in all time periods. This is mainly due to the price increase of Bitcoin Cash (CSH) which in addition to Dash is the only cryptocurrency that outperforms Bitcoin during the period. When Bitcoin's prices had surged in 2017-04-02, there seemed to be an overall breakout off value on multiple cryptocurrencies that did not mean revert towards the end of this period.

From the "Impact of trading action" charts we can conclude that 2h or the 1 day trading action created most value. (Figure 22)

Recent			Dynamic agent			Static agent			Equal weighted		
#	Period	Simulations	Ptf. value	MDD	Sharpe	Ptf. value	MDD	Sharpe	Ptf. value	MDD	Sharpe
31	15min	26	-11.84%	0.251	-2.6646	-9.55%	0.2569	-2.2848	-6.47%	0.2351	-1.6941
32	30min	32	-11.94%	0.2451	-2.6977	-10.26%	0.2526	-2.4341	-6.47%	0.2351	-1.6941
33	2h	30	-12.79%	0.2465	-2.8608	-11.32%	0.2568	-2.6364	-6.47%	0.2351	-1.6941
34	4h	24	-13.11%	0.2466	-2.875	-11.61%	0.2562	-2.6612	-6.47%	0.2351	-1.6941
35	1d	21	-12.6%	0.2126	-2.8824	-11.38%	0.2113	-2.7148	-6.47%	0.2351	-1.6941

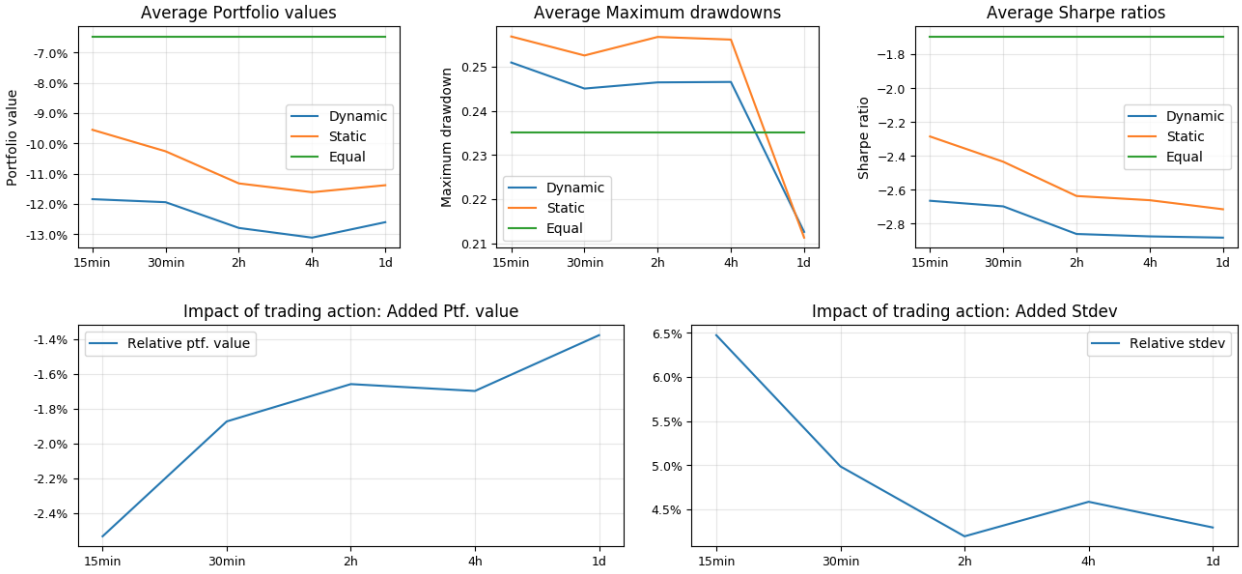


Figure 22: Recent - Aggregated report

Recent 15min

Strategy	Ptf value	Sharpe	Sharpe (ann.)	MDD
Dynamic agent	0.8819	-2.6681	-7.2087	0.2508
Static agent	0.905	-2.2896	-6.1861	0.2564
Equal weighted	0.9353	-1.6941	-4.5772	0.2351

Dataset	Start date	End date	Days	Steps
Train period	2018-07-01	2019-03-05	247	23820
Test period	2019-03-06	2019-04-26	50	4884

Parameter	Value
No. batches	20
No. episodes	3
Batch size	50
Trading fee	0.2%
Trading period	15min
Window length	70
Conv Filters, 1	5
Conv Filters, 2	50
Kernel size	(1, 3)
ε greedy thld	0.8

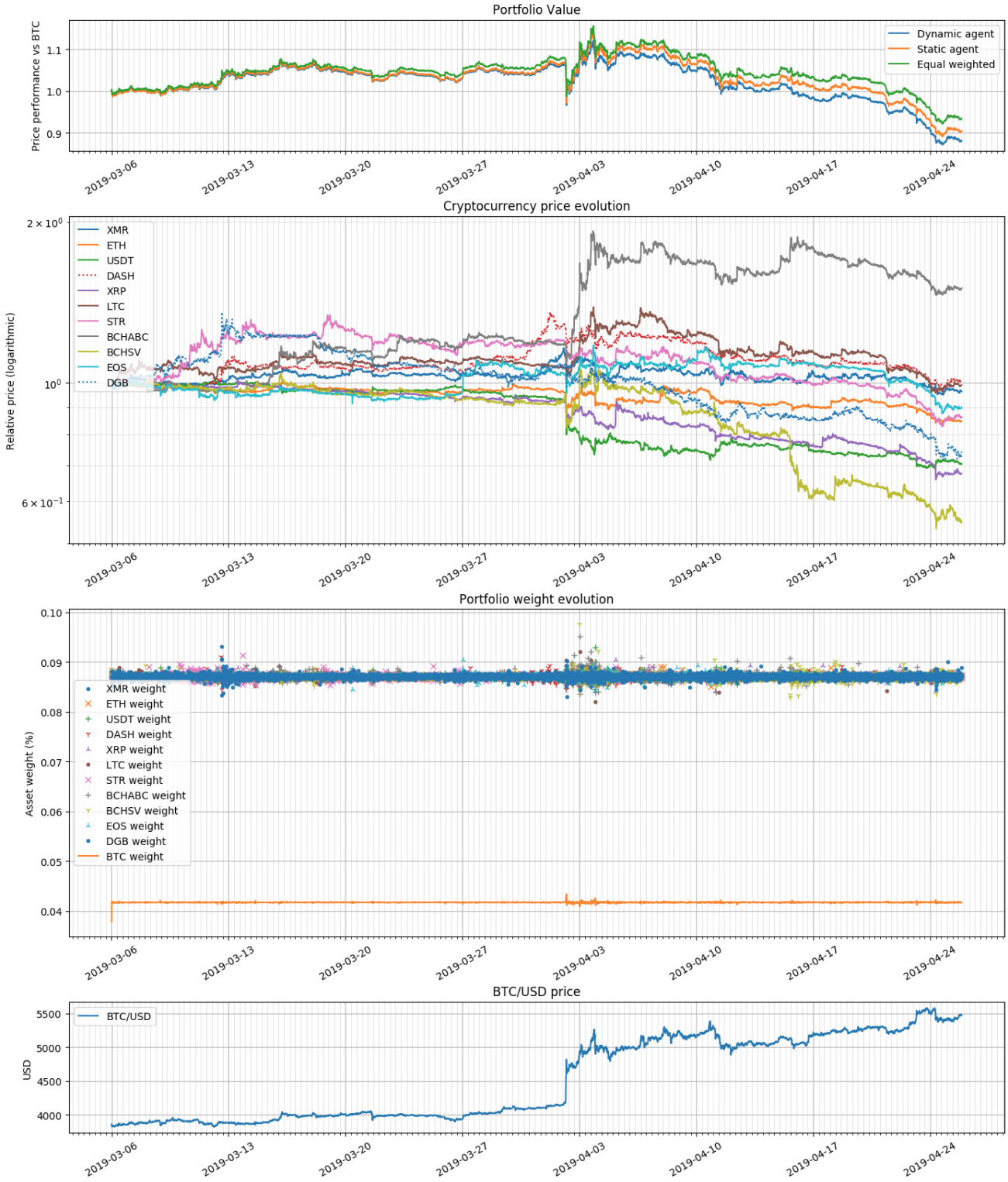


Figure 23: Recent - Simulation summary (15 min example)

5.7. Backtest summary

Above we demonstrated our trading agents in seven different market conditions. We have shown that the dynamic agent is a good tool for reducing uncertainty and it particularly excels in mean reverting conditions. This fact was demonstrated in backtests:

- Ripple bull run
- Ethereum valley
- Rock bottom

It was also able to significantly reduce volatility on the cost of portfolio value in “All-time high”.

In “Awakening”, were the magnitude of movements were low but frequent, the dynamic agent was able to create value at the cost of volatility.

In the backtest “Recent”, the dynamic agent did not work well as it betted on mean reversion when there was a fundamental breakthrough of prices.

The backtest “Calm before the storm” highlighted a fundamental risk of the dynamic agent: it continued to buy a plummeting asset (STEEM) while betting on mean reversion.

Overall, the dynamic agent performed best of the three strategies. In three backtests it was strictly the best and in two backtests it was able to either increase returns or decrease volatility.

However, the static agent loses to the equal weighted portfolio. This means that the neural networks predictions for optimal weight initializations destroys value. The loss was so high that it could not be accounted only for transaction cost. Therefore we could further enhance the dynamic agent by forcing it to initialize on equal weights.

The table on the next page summarizes the results (Table 4).

Table 4: Backtest summary

Backtest name	BTC price	BTC domc.	Price osc.	Ptf. values	MDDs	Sharpe ratios	Dynamic excess return	Dynamic excess volatility	Optimal trading period	Dynamic agent vs Static agent	Static agent vs Equal weight	Strategy ranks
Calm before the storm	Medium growth	Large growth	Low	-32% to -26%	0.29 to 0.34	-3.4 to -3.2	-4.6% to -3.2%	+6% to +11%	2h	Dynamic agent performs worse than static agent. All trading action destroys value and increases volatility	Static agent performs worse than equal weight due to smaller cash (BTC) weight	1. Equal weight 2. Static agent 3. Dynamic agent
Awakening	Large growth	Moderate decrease	Very low	+7% to +11%	0.14 to 0.19	1.0 to 1.4	-0.5% to +1.1%	+8% to +16%	30 min	Dynamic agent has a higher return on 15min to 4h. Also higher volatility but smaller MDD.	Static agent performs worse than equal weight due to smaller cash (BTC) weight	Equal for conservative, Dynamic for risk lovers
Ripple bullrun	No difference	Very large decrease	Very high	+142% to +170%	0.3 to 0.9	2.8 to 3.1	-0.5% to +7%	-2% to +11%	Not available	Dynamic outperforms static and equal agent on all indicators. A lot of mean reverting price action.	Static has higher return but also higher volatility.	1. Dynamic agent 2. Equal weight 3. Static
Ethereum valley	Medium growth	Slight increase	High	-3% to +4%	0.4 to 0.48	-0.1 to 0.48	+3% to +5.5%	-8% to -3%	30 min 2h	Large mean reversion profits for dynamic when Bitcoins dominance increases.	Static performs strictly worse than equal.	1. Dynamic agent 2. Equal weight 3. Static
All-time high	Extreme growth	Extreme decrease	Very high	+90% to +131%	0.4 to 0.8	2.45 to 2.65	-18% to -13%	-30% to -23%	30 min	Dynamic greatly loses in portfolio value but has much lower volatility. Either agent is arguably better.	Static outperforms equal due to decrease in bitcoin dominance	Dynamic for conservative, Static for risk lovers
Rock bottom	Very large decrease	Moderate decrease	Very low	+9% to +12.5%	0.1 to 0.22	2.3 to 3.3	+0.5% to +2.5%	-14% to -4%	2h 1 day	Dynamic thrives over static. There is a high amount of mean reverting price oscillation (e.g.: Bitcoin SV)	Static slightly loses to equal.	1. Dynamic agent 2. Equal weight 3. Static
Recent	Very large growth	Large growth	Very low	-13% to -6%	0.21 to 0.25	-2.8 to -1.7	-2.3% to -1.4%	+4.2% to +6.5%	2h 1 day	Trade action destroys value. This is due to price movement, that does not mean revert (for example Stellar (STR)	Static agent performs worse than equal weight due to smaller cash (BTC) weight	1. Equal weight 2. Static agent 3. Dynamic agent

5.8. Trading period analysis

One key goal of this study was to find out the optimal trading period length for cryptocurrency price optimization. Therefore each backtest was run on 5 different trading period lengths 15 min, 30 min 2 h, 4 h and 1 day. Since the static agent and the dynamic agent differ only by the rebalancing trading action made by the dynamic agent, we can compare the impact of trading action for each time period. In the table below (Table 5) we have collected the excess return and volatility created by the trading action.

We find evidence that the optimal trading period length for the dynamic agent is 30 min and 2 h. In the four backtest where the trading action was profitable, 30 min or 2 h rebalancing yielded the greatest benefit. Furthermore, even in the backtests where trading destroyed value 30 min and 2 h rebalancing destroyed the least amount of value. However, on the last two backtest, the performance of 1 day rebalancing was close to the 2 h rebalancing.

It would be interesting to find out the exact optimal period length of trading action for a given backtest. This could be probably reverse engineered by analyzing the price data.

It is quite surprising to find evidence that frequent rebalancing can create value. We suspect that this feature is unique to the cryptocurrency market. Furthermore, the decision to choose particularly exciting backtest periods might have affected this result. It would be interesting to try to understand this phenomena better by extending the analysis to other markets.

Table 5: Trading period analysis

Backtest name	Return	Volatility	Optimal trading period
Calm before the storm	-4.6% to -3.2%	+6% to + 11%	2h
Awakening	-0.5% to +1.1%	+8% to + 16%	30 min
Ripple bullrun	-0.5% to +7%	-2% to 11%	Not available
Ethereum valley	+3% to +5.5%	-8% to -3%	30 min, 2h
All-time high	-18% to -13%	-30% to -23%	30 min
Rock bottom	+0.5% to +2.5%	-14% to -4%	2h, 1 day
Recent	-2.3% to -1.4%	+4.2% to +6.5%	2h, 1 day

6. Discussion

Frankly, the strategy followed by the dynamic agent was disappointingly dull. Before building our model, we expected the agent to be able to find wildly oscillating patterns that would be profitable in the older backtest periods, but vanish in newer backtests. Finding the equally weighted behavior after building the model was quite underwhelming. As it stands, it would make sense to forget the fancy machine learning approach and build a lightweight trading algorithm that mimics the behavior of the dynamic agent. That kind of system would be much easier to understand, fine-tune and maintain.

However this study was not completely in vain. We did learn something new on equal weighted risk management in semi-high intervals. In this chapter we will first discuss, why the agent settled for equal weights instead of identifying more exotic trading strategies. Next we will discuss how the strategy learned by the dynamic agent could be utilized in practice and also in other asset classes than cryptocurrencies. Finally we will speculate why the dynamic agent was unable to learn any signals from the price data and how we could improve it with fuzzified representations of input data and some other adjustments.

6.1. Why the agent systematically chose equal weights

As the backtests showed, the strategy followed by the agent is very stationary. This was quite surprising for us since deep learning networks are known to be prone to overfitting: finding clear patterns in training data that fail to generalize in test data. We even experimented with very low transaction fees but even that had little to no influence on the strategy. It seems that the models fail to find any signals from the noisy raw price data.

One reason for this might be that we did not allow the deep learning to identify its tradable assets. In other words, we did not train a separate neural network for each asset in the portfolio but rather used the same one for all assets. Consequently, the neural network was unable to tune its parameters to longer term trends of some specific asset. Since the prices of cryptocurrencies in the portfolio were on average mean reverting, the model might have learned that on average the prices are mean reverting.

Another reason for the mean reverting behavior might be in how we framed the optimization question for the reinforcement learning part. If the question would have been to forget the

portfolio optimization part and just to train a deep learning network to predict which asset's price will increase or decrease on following periods, it might have found patterns. However, the question we had was not to detect price trends. Instead our goal was to build a model that changes weights in a portfolio to maximize returns. Due to transaction fees, this is quite a different question.

6.2. Risk management in semi-high frequencies

We learned from this study that an auto-balancing equally weighted strategy can outperform a static equally weighted strategy in highly volatile markets where there is mean reverting price action in short time intervals. The strategy could be useful in other than equal weight situations: with small tuning we could force the agent to start from predetermined weights. It would then guarantee that the relative weights would stay constant regardless of the price action in the market.

However, this type of strategy fails miserably if there is a fundamental change in the value of an asset. This was demonstrated in the "Recent" backtest, where Bitcoin Cash (BCH) stayed on a higher valuation to the end of the backtest. The dynamic agent lost to its static peers while it sold the majority of its position in BCH and the price continued to increase.

Even more worrying is the inverse situation as demonstrated in the "Calm before the storm" backtest. If the dynamic agent manages an asset whose fundamental value suddenly falls sharply, it would continue investing in it until it becomes effectively untradeable from the exchange's behalf. Therefore we would not recommend using this type of strategy without careful supervision.

The latter problem can be mitigated with two factors. Firstly, by holding a large amount of assets (+10) in the portfolio. In this case each individual asset only represents a small portion of the whole investment. The second and the more effective way would be to implement a stop loss mechanism that tracks how much the asset's absolute price has decreased from its original value and stops trading it altogether if its price goes below some predetermined threshold. This mechanism would be trivial to implement and append to any trading algorithm.

While we were able to demonstrate that the dynamic agent excelled in certain market conditions with surprisingly high frequencies, we do not expect that these findings would

generalize to other asset classes. Cryptocurrencies are notorious for being highly volatile. However, we expect that similar patterns could be found on some other volatile markets such as in equities in some emerging markets. Nevertheless, we suspect that profitability with the 30 minute rebalancing period is quite a unique feature in the cryptocurrency space.

A key weakness of this study was that we made the zero slippage and zero market impact assumptions. The dynamic agent's strategy would be tradable with small volumes, but in large amounts these two conditions would be violated. While these assumptions can be met by retail investors, these simplifications do not hold for institutional investors.

6.3. Improving the deep learning model

There are at least four things that we could do to improve the robustness of the deep learning model. Firstly, we could input it some technical indicators as input. Secondly, we could fuzzify the raw price data. Thirdly, we could utilize recurrent neural networks (RNNs). Finally, we could turn the problem from a regression task to an classification task.

The genetic algorithm approaches discussed briefly in the financial literature review part utilized technical indicators. In practice, the researchers had hand-picked some indicators such as exponential moving averages or Bollinger bands. The task of the algorithm was to find out meaningful combinations of these indicators that lead to optimal performance. This kind of approach could also be used with deep learning. It is a much easier task for the neural network to learn to find patterns from these kind of simplified feature than from raw data. However, it is dubious how well these kind of simplified features would generalize on unseen data and different market conditions.

A second way to improve the results would be to fuzzify the inputs similar to Deng et al. (2017). This is another way to reduce the variance in input data while not relying on technical indicators. In practice this would mean to reducing the raw price differences to corresponding rough degrees such as "very low", "low", "average", "high" or "very high". This kind of representation greatly reduces the dimensionality of the input data and is useful especially for continuous values.

A third way to improve the performance could be to use recurrent neural networks (RNNs) instead of convolutional neural networks. RNNs is another type of deep learning network that is widely used in practical applications such as Google translate. They are specifically designed to find patterns in sequential time series data. We did not use RNNs in this study for a few reasons: firstly Jiang et al.'s (2017) best performing models utilized CNNs instead of RNNs. Secondly, according to the author's previous experience RNNs are very slow to train. Thirdly, RNNs are designed to work in environments where the network can capture meaningful signals from a long sequence of time. Since financial data is noisy, we suspected that a pattern recognition network such as CNN would outperform an RNN.

The final way that the deep learning network could be altered would be to convert the regression problem into a classification problem. The actions of the DRL agent were defined as portfolio weights. This is in practice a vector of floating point numbers that sum up to one. The task of the agent was to figure out optimal floating point numbers from a continuous range between 0 and 1. Another way to design the actions would be to discretize them. The actions could be for example linguistic values like "Buy a lot", "Buy some", "Keep the same", "Sell some" etc. Behind each linguistic value could be a constant parameter such as "Buy some" = 1%. These parameters could also be trainable buy the network. This would effectively turn the portfolio management into a classification problem were these linguistic values are the classes. This would greatly reduce the dimensionality of choices for each asset in the output vector. Nevertheless, it would probably require considerable investigation to find out the optimal amount of output classes. Moreover, the generalization capability of this approach across backtest time periods and asset classes would be dubious.

7. Conclusions

This study explored two machine learning techniques in portfolio optimization: deep learning and reinforcement learning. We showed that these techniques converged into a profitable trading strategy, the auto-balancing equal weight. However, this strategy is not particularly unique or exciting. Nevertheless, the fact that the machine learning techniques converged into a profitable strategy is by itself already quite impressive. We did not code the agent to follow it. It found it by itself and followed it consistently. So in a way it did its job.

The strategy is especially useful in situations where the investor already holds a portfolio in a cryptocurrency market and expects that there will be significant short-term volatility. However, the investor cannot identify any particular coin that will increase or decrease in value. The strategy should be appended with a stop loss mechanism to avoid the risk of investing more money in a sinking ship as demonstrated in the "Calm before the storm" backtest.

To conclude, let's have a look at our research questions. Firstly, we wanted to find out whether the DRL agent is capable of identifying solid initial weights that will outperform the equal weighted strategy:

> **[H1] Successful weight initialization:** The static agent beats an equally weighted strategy on risk-adjusted return.

We find evidence against this hypothesis. On average, the static agent loses to the equally weighted strategy. This suggests that it cannot identify initial weights that outperform the equally weighted strategy.

Secondly we wanted to find out whether the trading action performed by the dynamic agent creates value:

> **[H2] Successful trading action:** The dynamic agent beats the static agent on risk-adjusted return.

We find some evidence for this hypothesis. In three of our 7 backtests the dynamic agent strictly beat the static agent: it had a higher return and a lower volatility. On two of the backtests it either reduced volatility or created considerable excess return. However, the two

remaining backtests demonstrated an important weakness of the dynamic agent. It loses to the static agent when there is a fundamental change in the price that remains to the end of the backtest period.

Our final research question was to find an optimal range of rebalancing period lengths:

> **[H3] Rebalancing period effect:** We are able to identify a range of trading period lengths that creates most value.

We find some evidence for this as 30 min and 2 h rebalancing period lengths generally outperformed. However, during the last two backtests the 1 day rebalancing performed as well as the 2 h. Therefore further research on longer rebalancing periods might be worthwhile.

This study made a contribution to the field of algorithmic finance and more specifically to the field of agent-based finance. To the best of our knowledge, this was the first study that explicitly demonstrates how a DRL agent balances its weights. We tested our trading agent in seven unique market conditions that demonstrated the agent's strengths and weaknesses. Furthermore, this was the first study in the cryptocurrency space that directly compared the impact of different rebalancing periods. The study raises some careful optimism that deep reinforcement learning is an useful technique in algorithmic finance. However, further investigation is needed to be able to generalize these results to other markets.

8. References

Aranha, C. and Iba, H. (2009): *"Using Memetic Algorithms To Improve Portfolio Performance In Static And Dynamic Trading Scenarios",* Proceedings of the 11th Annual conference on Genetic and evolutionary computation

Bhambhwani S., Delikouras S., Korniotis G. (2019): *"Do Fundamentals Drive Cryptocurrency Prices?"*
Available from: http://dx.doi.org/10.2139/ssrn.3342842

Bychkov, D. et al. (2017): *"Deep learning based tissue analysis predicts outcome in colorectal cancer",* Nature

Cortes, Corinna; Vapnik, Vladimir N. (1995): *"Support-vector networks"*, Machine Learning. 20 (3): 273–297.

Cybenko, G. (1989): *"Approximation by superpositions of a sigmoidal function",* Mathematics of Control, Signals and Systems, December 1989, Volume 2, Issue 4, pp 303–314

Deng, Y. et al. (2016): *"Deep Direct Reinforcement Learning for Financial Signal",* IEEE Transactions on Neural Networks and Learning Systems (Volume: 28 , Issue: 3 , March 2017)

Goodfellow, I. (2016): "*Deep learning*", The MIT press
Available from: https://www.deeplearningbook.org/

Gorgulho, A., Neves, R. and Horta, N. (2011): *"Applying a GA kernel on optimizing technical analysis rules for stock picking and portfolio composition"*, Expert Systems with Applications

Ha, S., Moon,. (2017): *"Finding attractive technical patterns in cryptocurrency markets",* Memetic Computing
Available from: https://link.springer.com/article/10.1007/s12293-018-0252-y

Heaton, J., Polson, N. and Witte, J. (2016): *"Deep learning for finance: deep portfolios"*, Applied Stochastic Models in Business and Industry

Holland, J.H. (1975): *"Adaptation in Natural and Artificial Systems"*, University of Michigan Press

Hornik, K., Stinchcombe, M., White, H. (1989): *"Multilayer feedforward networks are universal approximators",* Neural Networks, Volume 2, Issue 5, 1989, Pages 359-366

Hubel, D. H. and Wiesel T.N, (1968): *"Receptive fields and functional architecture of monkey striate cortex"*

Jiang, Z., Xu, D., and Liang, J. (2017): *" A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem"*, arXiv.org
Available from: https://arxiv.org/abs/1706.10059

Kim, K., Han, I. (2000): *"Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index",* Expert Systems with Applications

LeCun, Y et al. (1989): *"Backpropagation Applied to Handwritten Zip Code Recognition"* Neural Computation (Volume: 1 , Issue: 4 , Dec. 1989)
Available from: https://www.mitpressjournals.org/doi/abs/10.1162/neco.1989.1.4.541

Lin, C and Liu, Y (2008): *"Genetic algorithms for portfolio selection problems with minimum transaction lots"*, European Journal of Operational Research

Lo, A. (2003): "*The Statistics of Sharpe Ratios",* Financial Analysts Journal 58(4)

Markowitz, H.M. (March 1952). *"Portfolio Selection".* The Journal of Finance

McCulloch, W., Pitts, W. (1943): *"A logical calculus of the ideas immanent in nervous activity"*, The bulletin of mathematical biophysics

Nakamoto, S. (2008): *"Bitcoin: A Peer-to-Peer Electronic Cash System"*

Rosenblatt, F (1958): *"The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain"*, Psychological Review, v65, No. 6, pp. 386–408.

Silver, D. et al. (2016): *"Mastering the game of Go",* Nature
Available from: http://augmentingcognition.com/assets/Silver2016a.pdf

Tesauro, G. (1995) : *"Temporal Difference Learning and TD-Gammon"*, Communications of the ACM 38(3):58-68

Yan W., Sewell M. and Clack C. (2008): *"Learning to Optimize Profits Beats Predicting Returns — Comparing Techniques for Financial Portfolio Optimization",* Proceedings of the 10th annual conference on Genetic and evolutionary computation

Waltz (1965): *"A heuristic approach to reinforcement learning control systems"*
Available from: https://ieeexplore.ieee.org/abstract/document/1098193

Wu Yonghui et al. (2016): *"Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation"*
Available from: https://arxiv.org/pdf/1609.08144.pdf

Yan W., Sewell M. and Clack C. (2008): *"Learning to Optimize Profits Beats Predicting Returns — Comparing Techniques for Financial Portfolio Optimization",* Proceedings of the 10th annual conference on Genetic and evolutionary computation

9. Appendices

9.1. Additional material on Deep learning

9.1.1. Differences between biological and artificial neural networks

Before dwelling deeper into deep learning theory, we would like to discuss the differences between biological and artificial neural networks. Human neural networks and artificial neural networks differ significantly at least for the following factors:

ANNs and human neural networks have different structure. Neurons in human brains can pretty much be wired to any other arbitrary neuron forming complex circular patterns and feedback loops. Contrarily, ANNs have a rigorous hierarchical structure that allows only one directional signal sending.

A second important difference is in the timings when neurons are allowed to fire. Biological neurons can fire asynchronously at any given moment, regardless of the firing of other neurons. ANNs must wait that the previous layer has been completely processed before moving on to the next layer.

Thirdly, human neurons use different kinds of activation functions which is the function that decides whether the neuron should fire or not. Activation functions will be discussed in detail in a later section.

The human brain has about 100 billion neurons. This huge amount of neurons is well beyond the amount that a modern computer can hold in memory.

9.1.2. Deep learning revolution of the 2010s

While ANNs are one of the first AI research subjects (McCulloch, Pitts, 1943), they have only recently caught mainstream adoption. Their huge increase in popularity is due to at least the following factors:

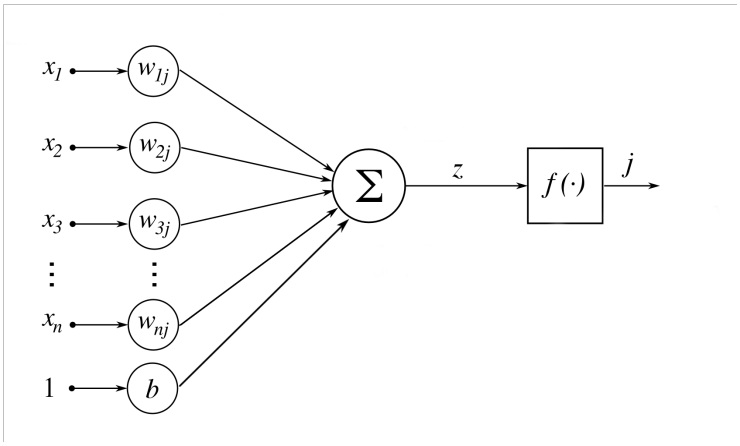
Firstly, deep learning models require a significant amount of data to outperform other machine learning models (Goodfellow, 2016). Modern digitized society where everything is connected to the internet has laid the foundation for more data intensive methods to excel .

Secondly, training a large DL network requires a huge amount of computational resources. Training an ANN is an iterative process of solving linear equations, computing gradients and optimizing parameters (Goodfellow, 2016). While there is an immense amount of individual computations required to optimize the network, each individual computation is relatively light. Since the individual computations can be performed concurrently, Graphics processing units (GPUs) are optimal for training DL models. (Goodfellow, 2016) Serendipitously, the computer gaming industry subsidized the development of the relevant technology needed for DL by developing powerful GPUs during the beginning of the millennia.

Thirdly, DL has been democratized to a larger base of developers. Nowadays there exists many open-source DL frameworks such as TensorFlow, Keras and PyTorch. These frameworks compete in various aspects such as ease of use, training speed, scalability to multiple devices to name a few. A powerful DL model can be developed with all of the frameworks mentioned above with only a few lines of code. Furthermore, the default parameters of all of the frameworks are aligned with what is considered as best practice in DL literature. Before the existence of these frameworks, a developer needed to be an expert in deep learning literature, calculus, linear algebra and efficient computation even to implement a simple neural network.

9.1.3. Artificial neuron

A deep learning model is a based on a huge amount of simple units called artificial neurons. Artificial neurons are based on the perceptron algorithm (Rosenblatt, 1958). A single neuron is basically a function that takes a vector of signals ***x*** as an input and outputs a non-linear transformation *j* of the weighted sum of the inputs. Let's dissect this step by step:



The first step of the artificial neuron is to sum the signal vector ***x*** to produce *z*. Usually the signal x_i is weighted by some weight w_{ij} where *j* corresponds to the *jth* layer of the neural network. The weights of the neural network are held by a matrix ***W*.**

In addition to the weights, each artificial neuron holds a bias term *b* as a trainable parameter**.** It is analogous to the intercept term in linear regression: it basically tells how large the weighted sum of input signals ***x*** and weights ***w*** need to be before the neuron becomes meaningfully active.

Our final weighted *z* with bias included can be calculated as dot product of weights and activations:

$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \\ b \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ 1 \end{bmatrix} = \mathbf{w^T x} = \mathbf{z}$$

The final step of the artificial neuron is to feed the *z* to some kind of *activation function*. The output of the artificial neuron, *j*, is the scalar value produced by the activation function.

Why do we need an activation function *f*? *The activation function is necessary for adding nonlinearity to our Deep learning model.* We will eventually implement a deep learning

model that learns to approximate an arbitrary nonlinear statistical problem with a network of hierarchical concepts. The activation function guarantees that the change of the output of the model is not proportional to the change of the input.

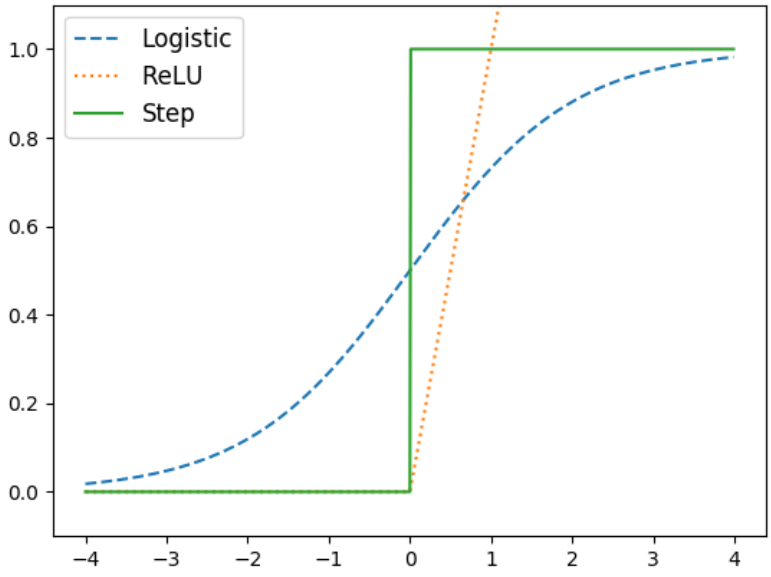
Without nonlinearity, the output can be represented a linear combination of the input. The hidden layers are redundant based on linear algebra. A nonlinear activation function is an requirement for the Universal approximation theorem to hold. Without it, the MLP is not capable to approximate a nonlinear function. (Hornik 1991, Cybenko 1989).

Below are some examples of common nonlinear functions:

$$Step(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$Logistic(x) = \frac{1}{1 + e^{-x}}$$

$$ReLU(x) = max(0, x)$$



All of the three functions satisfy the nonlinear property of not being proportional to the size of their input.

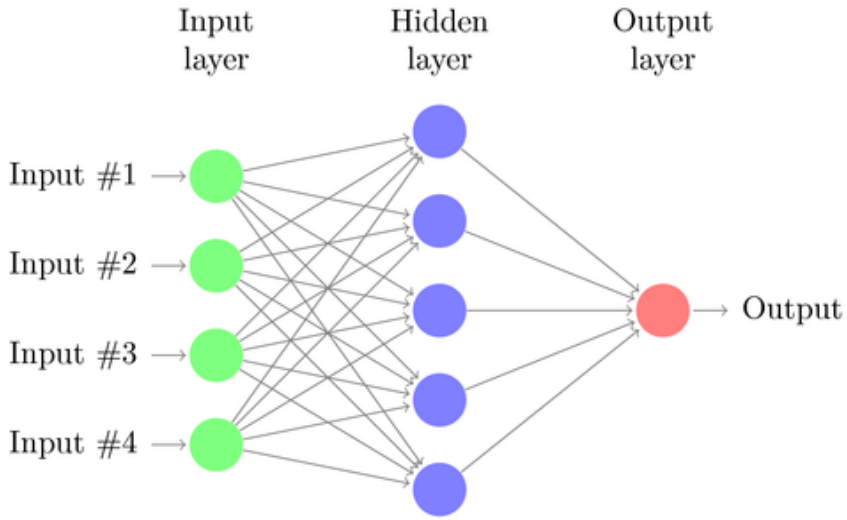
- For positive inputs, the step function returns 1 regardless of how large the input is.
- The logistic function is a softened version of the step function, but likewise slowly converges to 1 when the size of the input grows.
- The ReLU function is nonlinear because it cannot take values below zero.

In this study we will utilize the Logistic and variants of the ReLU function. They both have a nice property of being differentiable.

9.1.4. Multilayer perceptrons (MLPs)

The classic implementation of a DL model is called a multilayer perceptron (MLP). This type of network consists of multiple layers of artificial neurons stacked on top of each other.

The first and last layers of an MLP are special. The first layer (i.e.: the input layer) consists of the dependent variables. For example, if the task is a financial prediction problem, the input layer could consist of a vector, ***x,*** of the price differences of stocks. The last layer (i.e.: the output layer) contains the predictions/ output vector of the MLP $\hat{\boldsymbol{y}}$. The vector $\hat{\boldsymbol{y}}$ could contain for example a vector of predicted price differences for the next time period.



Each connection between artificial neurons transmits signals. The neuron that receives the signal can either choose to ignore or process a signal. After a neuron has processed every signal it has received, it fires/ sends a new signal forward to the network. The connections between the artificial neurons are known as *weights* and are almost always represented by real numbers. A larger weight implies a stronger connection between the artificial neurons.

The goal of an MLP is to approximate some highly nonlinear function $\boldsymbol{g}$ that maps some input vector $\boldsymbol{x}$ to a desired output $\boldsymbol{y}$:

$$y = g(x)$$

For the underlying function $\boldsymbol{g}$ we want to estimate some kind of approximator $\boldsymbol{g_{approx}}$ that has a set of parameters $\boldsymbol{\theta}$ that we can optimize. According to the Universal approximation theorem, a MLP is fully capable of approximating any nonlinear function (Hornik 1991, Cybenko 1989).

$$\hat{y} = g_{approx}(x, \theta)$$

We would like to train our approximator of the function ***g*** to be as close as possible to the underlying function. This will be achieved by training the network's parameters ***θ*** as discussed in a further section. Now let's have a closer look at the function ***g*** itself:

The approximator for the function ***g*** can be denoted as a chain of functions where each layer of the network is represented by its own function:

$$\hat{y} = g(x) = h(i(x))$$

, where each layer has its own set of parameters ***θ:***

$$\hat{y} = h(i(x, \theta_i), \theta_h)$$

Furthermore, each neuron in a layer can be viewed as its own function: each weight it receives is a parameter and its output is the signal it fires. Therefore *an MLP can be viewed as an collection of simple mini functions that interact at a massive scale.*

The MLP has three important properties:

Feed forward: An artificial neuron in a specific layer can only receive input from the previous layer and fire an signal to the next layer.

Fully connected: Each neuron is connected by weights to every neuron in the previous layer and to every neuron in the next layer.

1+ hidden layers: An MLP has at least one hidden layer in between input and output layers. These hidden layers enable the MLP to break the problem of mapping inputs to outputs into a hierarchy of concepts. The lower level concepts are represented in lower layers of the network (i.e.: close to the input layer). The network then learns to combine these lower layer concepts in higher layers of the network (i.e.: close to the output layer) (Goodfellow, 2016). For this reason, a funnel shaped MLP usually works pretty well: a large collection of lower layer concepts are gradually narrowed to higher level concepts.

The layered architecture gives the name deep learning: there is not just a single layer of neurons but multiple. The amount of hidden layers in the MLP determines the depth of the network. Deep MLPs can have dozens of layers.

There are two important algorithms related to MLPs:

1) **Forward propagation:** The process the MLP uses to make predictions.

2) **MLP learning algorithm:** The process training the MLP parameters θ.

We will tackle both of these algorithms next.

9.1.4.1. Forward propagation, making predictions

The goal of an MLP is to discover patterns from a large amount of training samples and be able to generalize to new samples. Since the underlying unknown relation between the inputs and outputs is probably highly nonlinear, we need to come up with a nonlinear way to make predictions.

The process of feeding the MLP an input vector $\boldsymbol{x}$ and receiving an output $\hat{\boldsymbol{y}}$ is called **forward propagation** (Goodfellow, 2016). Let's say you have an MLP with N layers. You also have some input vector $\boldsymbol{x}$ and a weight matrix $\boldsymbol{W}$, that describes the connection weights between neurons. The process is the following.

1. You place an input vector, $\boldsymbol{x}$ to the input layer**.**
2. For each node i in the input layer, a signal x_i is sent forward to the network. Every neuron, $h_{n,i}$ in the first hidden layer receives this signal.
3. For each hidden neuron, $h_{n,i}$, all the received signals, x_i, are multiplied with the signal weight $w_{n,i}$. The weighted signals are then summed up and fed to an activation function. The neuron's output $j_{n,i}$ is equal to the output of the activation function.

$$f\left(\begin{bmatrix} w_{n,1} \\ w_{n,2} \\ \vdots \\ w_{n,i} \\ b \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ 1 \end{bmatrix}\right) = f\left(\mathbf{w_n^T x}\right) = f\left(\mathbf{z_{n,i}}\right) = j_{n,i}$$

4. For each hidden unit, $h_{n,i}$, we pass the final output $j_{n,i}$, forward to the next layer. If the next layer is a hidden layer, perform step 3 again. If the next layer is the output layer, move on to the next step.
5. The output layer contains a vector of the models predictions $\hat{\boldsymbol{y}}$. Depending on the task, this layer usually has some activation function for example the softmax function.

Why do we need an activation function for the output layer? Let's say we want to implement an MLP that optimizes an financial portfolio. Our portfolio consists of *K* assets. For each asset, k_i, we would like to estimate an optimal portfolio weight to maximize return. Each of these weights need to be between zero and one (ignoring short selling here). Furthermore, the sum of these weights need to sum to one.

Our MLP's output layer in this case would have *K* neurons, one for each asset k_i. We need to ensure that the outputs of the neural network k_i sum up to one. For this purpose we will add an activation function to the output layer called the *softmax function* (or normalized exponential):

$$softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Unlike other activation functions we have looked at, the softmax function takes all of the outputs of the layer as an input and normalizes them based on *e*. This guarantees that the weights of our output layer sum up to one.

The quality of the MLP's predictions $\hat{\boldsymbol{y}}$, is dependent on how well the weights of the MLP are tuned to capture relevant signals from the input. In the next section we will describe how these weights are actually trained and optimized.

9.1.4.2. Training the network

We would like to optimize the MLP's parameters θ, to approximate an arbitrary nonlinear function. The parameters are at the beginning of the training process initialized to some random values. Initially the MLP will perform terribly and these random values need to be optimized to be able to capture signals from the input data.

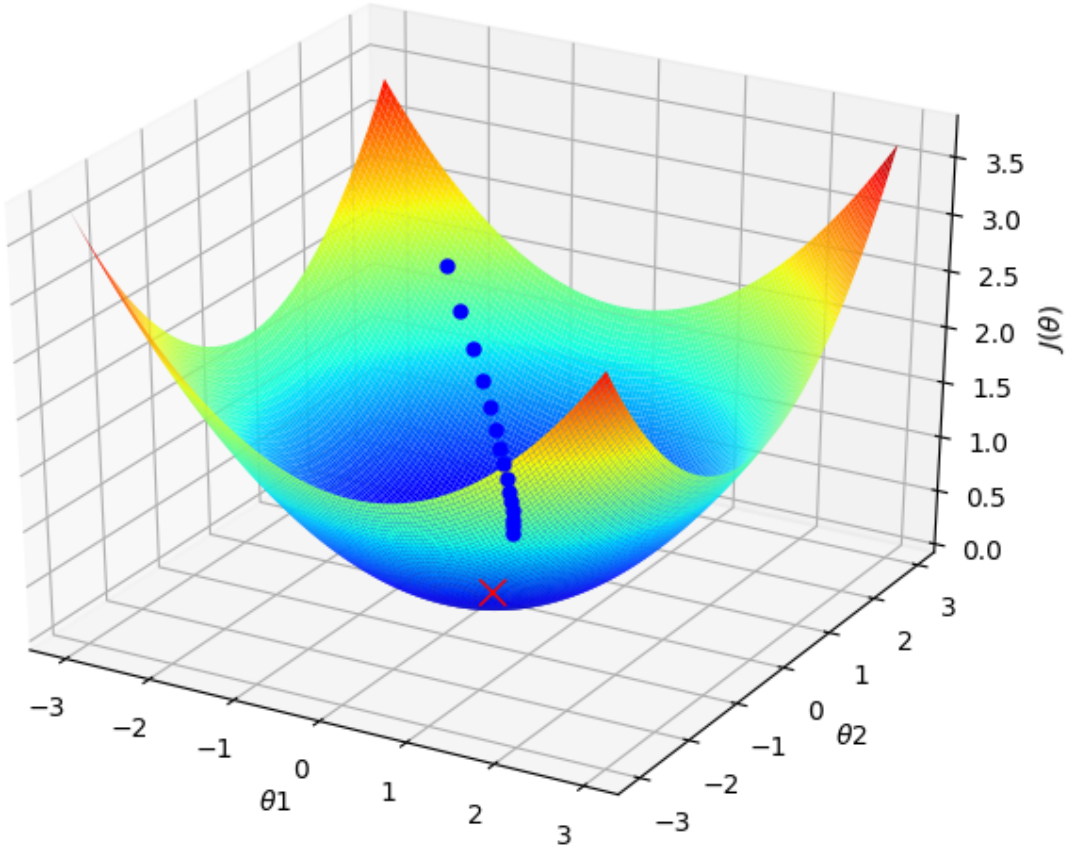
We want to train the MLP by first measuring how each individual parameter contributed to the performance of the model and then nudging these parameters to a slightly more optimal setting. This is achieved in three steps (Goodfellow, 2016):

1. **Cost function:** The performance of the model is evaluated by calculating a cost vector ***J(θ)***.
2. **Back-propagation:** The contribution of each parameters impact θ_i to the cost ***J(θ)*** is collected to a gradient vector $\hat{\boldsymbol{g}}$.

3. **Gradient descent:** The gradient vector $\hat{\boldsymbol{g}}$ is used to optimize parameters $\boldsymbol{\theta}$.

Before going into details regarding each of the three steps, let's have a look at an simplified version of the problem where there are only two parameters, θ_1 and θ_2.

In the z-axes we have the cost $J(\theta)$ and the plot visualizes how the cost changes when parameters θ_1 and θ_2 changes. The parameters, θ_1 and θ_2 are initially set to some random values which is visualized by the highest blue circle. We iteratively find the direction of the steepest descent and slightly update our parameters toward that direction as visualized by the path of blue circles. The same principle holds when we have n parameters instead of only 2.



Step 1: Cost function

We want to calculate a cost vector ***J(θ)*** that describes the performance of our neural network. The cost describes how close our prediction vector, $\hat{\boldsymbol{y}}$ is to the true values $\boldsymbol{y}$. (Naturally, this implies that the training process of an MLP requires a labeled training set where the true values of $\boldsymbol{y}$ are known.) We will utilize different cost functions based on whether the task is discrete or continuous.

In **discrete tasks** the **cross-entropy cost function** is commonly used. In a discrete problem we want to train the neural network to correctly classify an training instance $\boldsymbol{x_i}$ vector to some true target class *k.* The output layer of the neural networks contains *K* neurons, one for each target class *k*. At the end of the forward propagation step, the output layer contains probability vector $\hat{\boldsymbol{y}}$**,** that describes how likely it is that the instance $\boldsymbol{x_i}$ belongs to an individual class $\hat{y}_{i,k}$.

Let's say we have *m* training examples and *K* target classes. The cost ***J(θ)*** is calculated by computing the average distance between the true class vector $\boldsymbol{y}$ and the estimated probabilities $\hat{\boldsymbol{y}}$**,** across all training examples *m.*

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_{i,k} \times ln(\hat{y}_{i,k})$$

In **continuous tasks** the **mean squared error cost function** is commonly employed. Since we are trying to estimate a single continuous value $\hat{y}_i$ for our training vector $\boldsymbol{x_i}$, we only need a single neuron in the output layer. The mean squared error describes how far a prediction $\hat{y}_i$ is from the true value y_i on average across training examples *m*:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

Step 2: Back-propagation

The task of the back-propagation algorithm is to compute the gradient $\hat{\boldsymbol{g}}$ (i.e.: a vector of all partial derivatives) of the neural networks parameters regards to the cost ***J(θ)***. This is achieved by exploiting the chain rule of calculus to iteratively calculate the contributions of each neurons weights to the cost. (Rumelhart et al., 1986)

The back-propagation takes as an input the cost vector ***J(θ)*** that describes the average error in each output neuron. The backpropagation uses this cost vector ***J(θ)*** and computes how much each neuron in the last hidden layer contributed to each element in the cost vector.

Then it recursively computes the error contribution of the neurons in the previous hidden layer by utilizing the chain rule of calculus:

$$F'(x) = f'(g(x))g'(x)$$

Chain rule in Liebniz notation:

$$\frac{dz}{dx} = \frac{dz}{dy} \times \frac{dy}{dx}$$

Eventually the process reaches the input layer and the contribution of all parameters in the MLP to the cost vector ***J(θ)*** has been computed. The contributions are stored to the gradient vector $\hat{\boldsymbol{g}}$ and passed to the gradient descent step. For more details read Rumelhart.

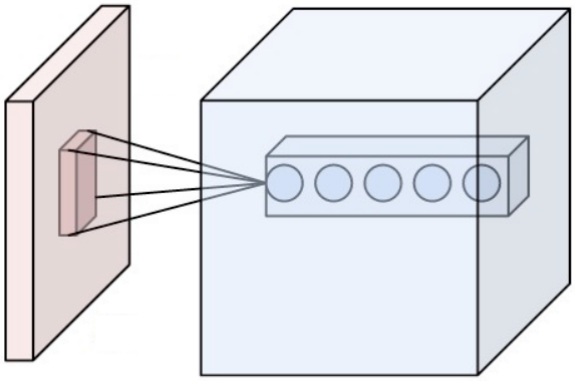
Step 3: Gradient descent

Once the gradient $\hat{g}$ of the cost has been calculated, we can use an optimization algorithm such as stochastic gradient descent to slightly optimize the weights of the MLP:

$$\theta_{k+1} = \theta_k - \lambda \hat{g}_k$$

,where λ is the learning rate of the algorithm (a small constant). This simple gradient descent algorithm is usually replaced by a more powerful variant in practical applications. Some common variants include the Adam optimizer or the Momentum optimizer.

9.1.5. Convolutional neural networks (CNNs)



In practical applications, classic MLPs are almost always combined with other neural network architectures. One of the most popular architectures are Convolutional neural networks (LeCun et al., 1989). CNNs were originally designed for image data and have proven to be highly successful in a wide range of computer vision tasks. However, the CNN architecture has proven to be very general: it excels in any environment where the input data has a grid-like structure. This includes financial time-series data, where the time-series data can be viewed as a 1D grid of sequential samples (Goodfellow, 2016).

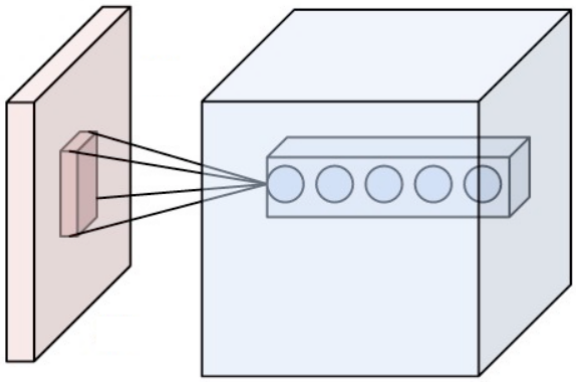
CNNs are inspired by the functions of the animal visual cortex. Similar to the visual cortex, neurons in a convolutional layer are only affected by previous neurons that are in their receptive field. (Hubel et al., 1968)

CNNs outperform MLPs because they enable far larger networks to be trained while having considerably less parameters to train. This is possible, because the convolutional layers are only *partially connected to the input.* Recall that in an fully connected MLP each input neuron is connected to each hidden neuron in the next layer. This means that the amount of connections *grows exponentially* when the amount of inputs or neurons in the hidden layer increase.

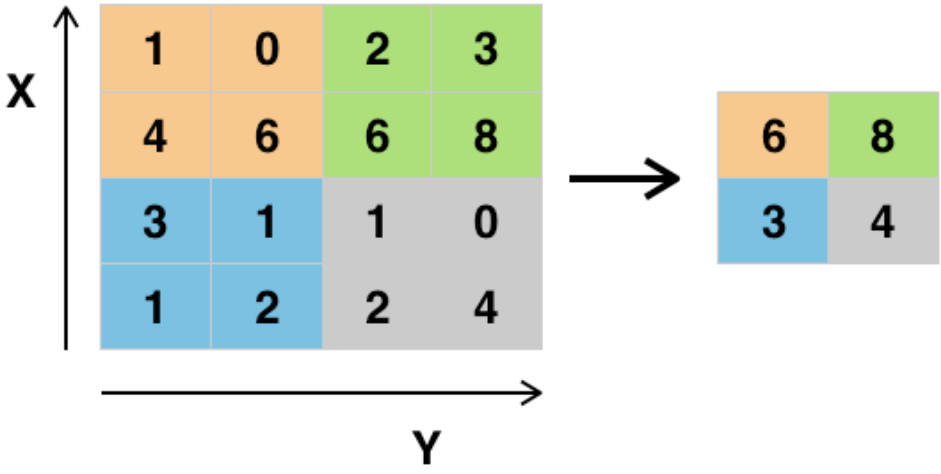
Convolutional layers introduce parameter efficiency by employing small filters to the input layer. An individual convolutional layer can have dozens or even hundreds of filters. Since a single filter shares the same parameters, the computational load of adding a filter is *linear* to the number of filters. The same small filter is applied to the whole input step by step. This reduces the amount of parameters of the model significantly.

The convolution process also has the important property of generalizing its findings of patterns to the whole input grid. This ensures that if a pattern is recognized in one part of the input grid, it can also be recognized in any other part. On the contrary, a MLP can only learn to recognize patterns in particular locations of the input grid.

Below is an visualization of the convolution process. The convolutional layer (blue box) employs five filters (no. of blue circles) to the 2D input grid. The same five filters are applied to each area of the 2D input grid.



CNNs almost always employ an pooling operation right after the convolutional layer. The point of the pooling layer is to remove unnecessary noise from the network (i.e.: reduce dimensionality). The most common pooling function is the max pooling function which works by taking the maximum value from its receptive field:



A typical CNN architecture usually has a set of convolutional layers at the lower layers (with max pooling layers in between). These convolutional layers are then followed by fully connected layers. The intuition here is that the convolutional layers learn a better

representation of the input and the fully connected layers use this representation to tackle the statistical prediction problem. The CNN architecture will be used in this study.

9.2. Stability reports

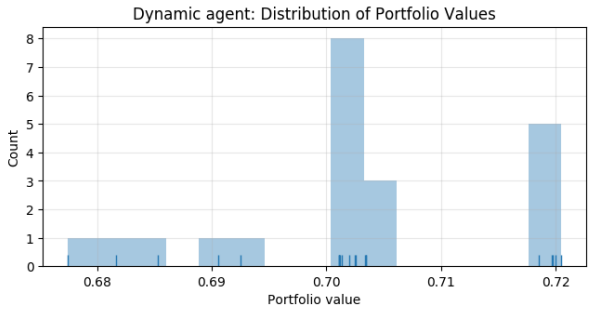
[Simulation stability] Calm before the storm 15min

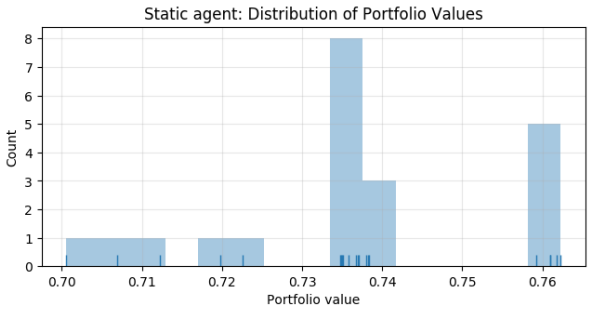
Dynamic agent	Average	Stdev
Ptf. value	0.7024	0.0121
Sharpe ratio	-3.378	0.0025
Sharpe ratio (ann)	-9.1269	0.0068
MDD	0.326	0.0115
Average of weights	0.0866	0.002
Stdev of weights	0.0003	0.0
Cash weight (BTC)	0.0469	0.0222

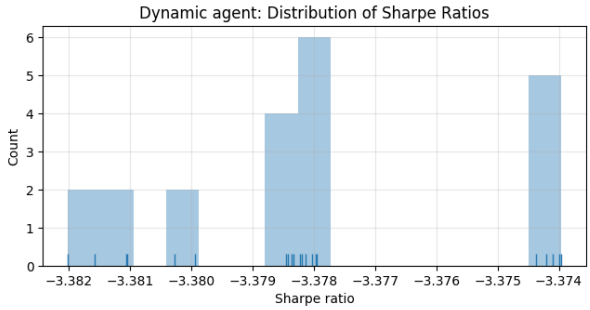
Static agent	Average	Stdev
Ptf. value	0.7366	0.0173
Sharpe ratio	-3.3151	0.0104
Sharpe ratio (ann)	-8.957	0.0281
MDD	0.2925	0.0167
Average of weights	0.0866	0.002
Stdev of weights	0.0003	0.0
Cash weight (BTC)	0.0469	0.0222

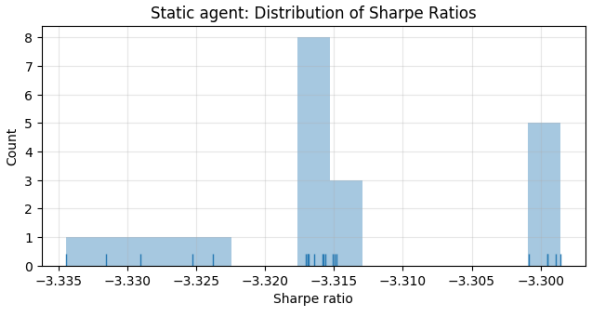
Equal weighted	Average
Ptf. value	0.7676
Sharpe ratio	-3.2455
Sharpe ratio (ann)	-8.7687
MDD	0.2661

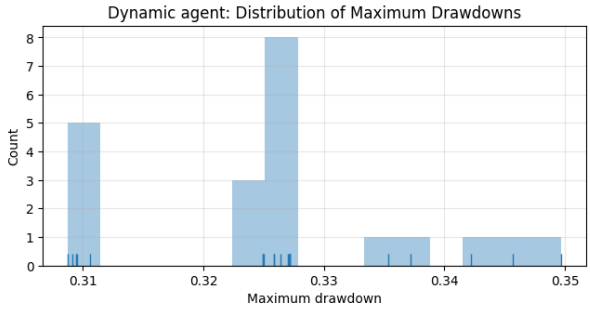
Parameter	Value
No. of simulations	21
No. of assets	11
Start date	2016-09-07
End date	2016-10-28
Trading period	15min

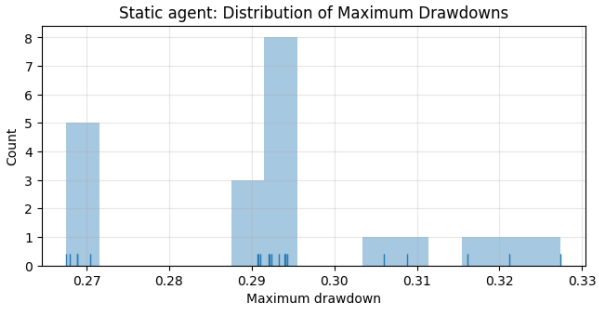


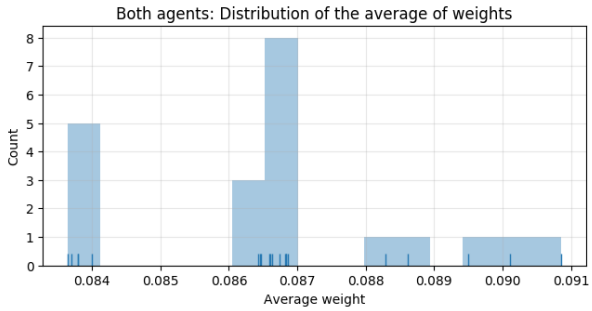


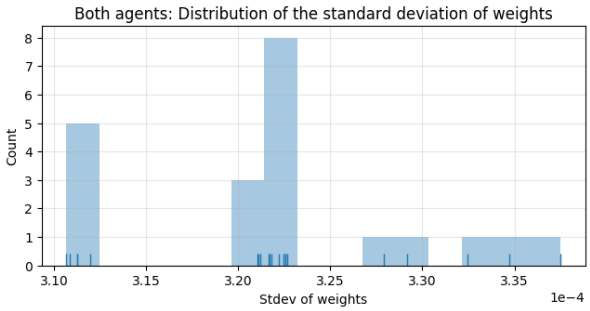












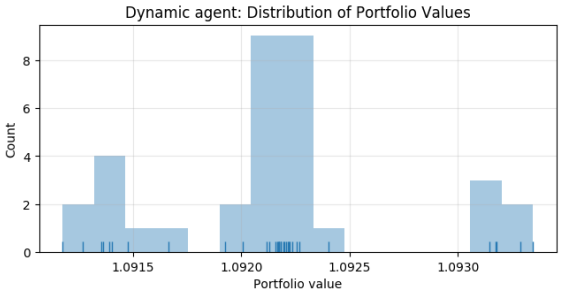
[Simulation stability] Awakening 2h

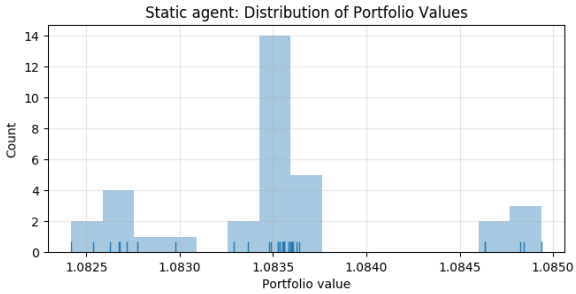
Dynamic agent	Average	Stdev
Ptf. value	1.0921	0.0006
Sharpe ratio	1.1896	0.0111
Sharpe ratio (ann)	3.214	0.0299
MDD	0.158	0.0029
Average of weights	0.0869	0.0017
Stdev of weights	0.0027	0.0001
Cash weight (BTC)	0.0441	0.0182

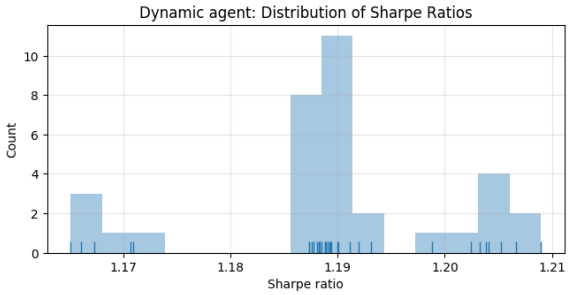
Static agent	Average	Stdev
Ptf. value	1.0835	0.0006
Sharpe ratio	1.2238	0.0105
Sharpe ratio (ann)	3.3065	0.0285
MDD	0.1838	0.0036
Average of weights	0.0869	0.0017
Stdev of weights	0.0027	0.0001
Cash weight (BTC)	0.0441	0.0182

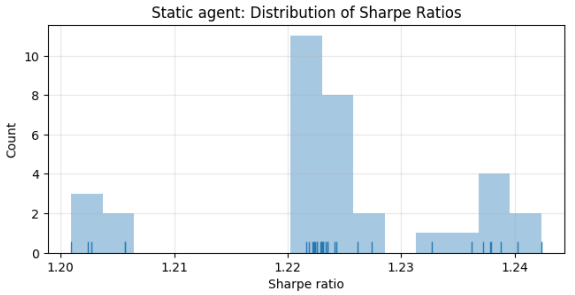
Equal weighted	Average
Ptf. value	1.0861
Sharpe ratio	1.3021
Sharpe ratio (ann)	3.5181
MDD	0.1769

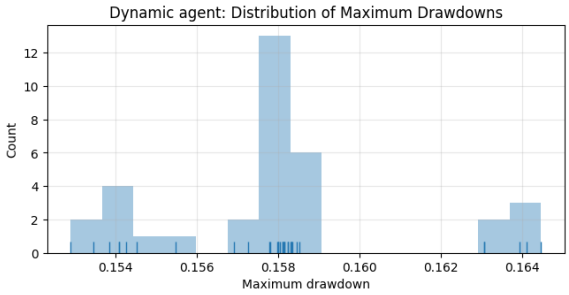
Parameter	Value
No. of simulations	34
No. of assets	11
Start date	2016-12-08
End date	2017-01-28
Trading period	2h

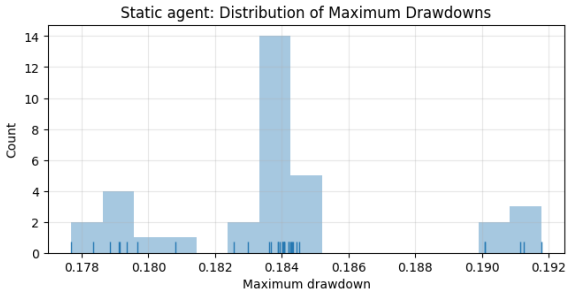


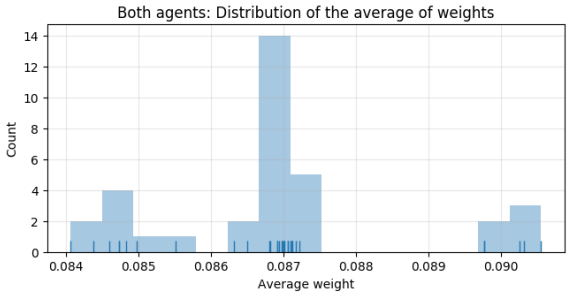


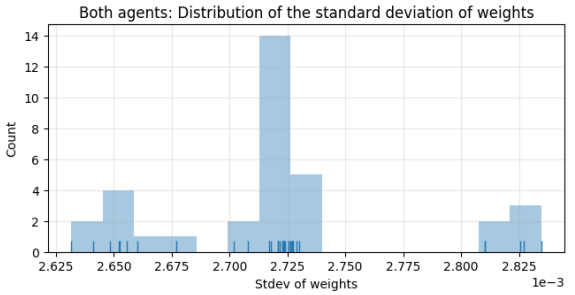












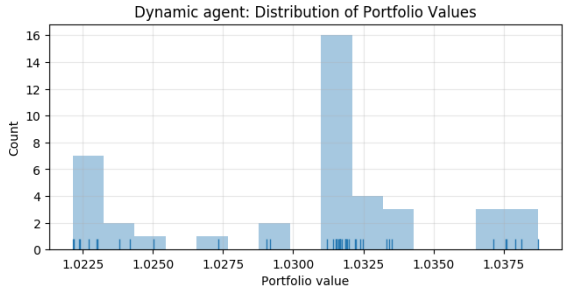
[Simulation stability] Ethereum valley 30min

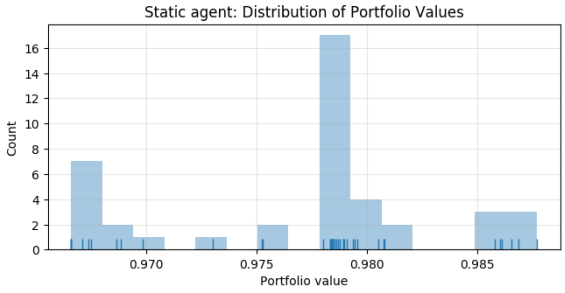
Dynamic agent	Average	Stdev
Ptf. value	1.0305	0.0048
Sharpe ratio	0.3407	0.0629
Sharpe ratio (ann)	0.9205	0.1699
MDD	0.479	0.0144
Average of weights	0.0875	0.0022
Stdev of weights	0.0012	0.0
Cash weight (BTC)	0.038	0.0247

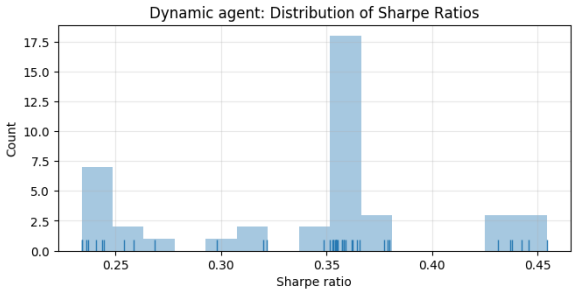
Static agent	Average	Stdev
Ptf. value	0.977	0.0061
Sharpe ratio	-0.2358	0.0547
Sharpe ratio (ann)	-0.6372	0.1477
MDD	0.4681	0.0143
Average of weights	0.0875	0.0022
Stdev of weights	0.0012	0.0
Cash weight (BTC)	0.038	0.0247

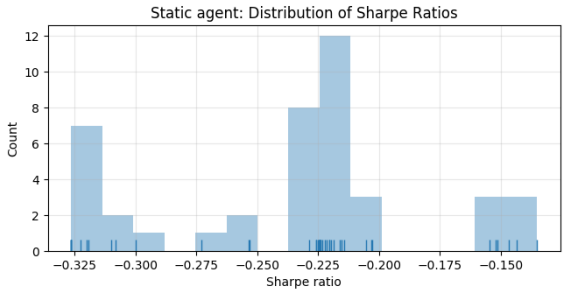
Equal weighted	Average
Ptf. value	0.9918
Sharpe ratio	-0.0898
Sharpe ratio (ann)	-0.2427
MDD	0.4434

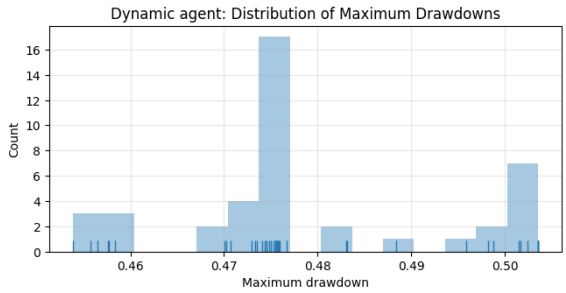
Parameter	Value
No. of simulations	42
No. of assets	11
Start date	2017-05-28
End date	2017-07-18
Trading period	30min

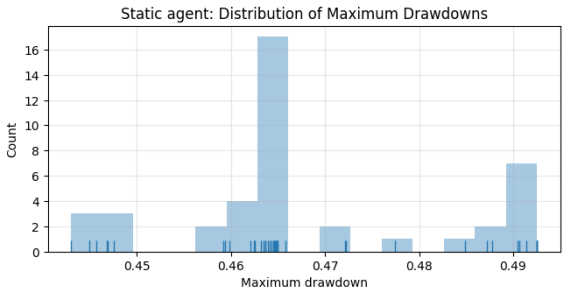


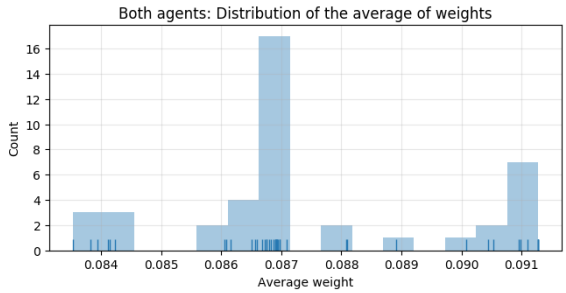


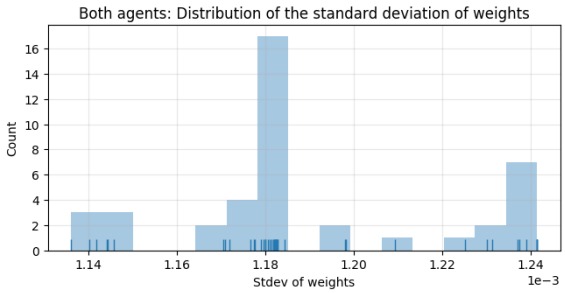












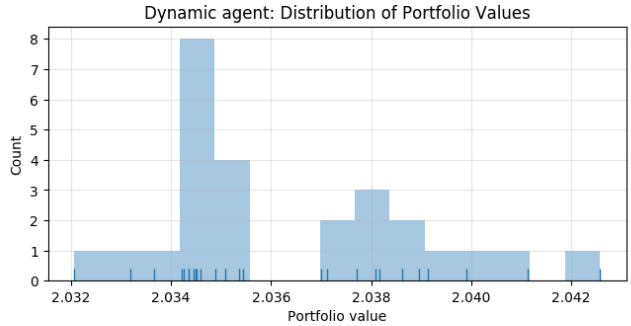
[Simulation summary] All-time high 15min

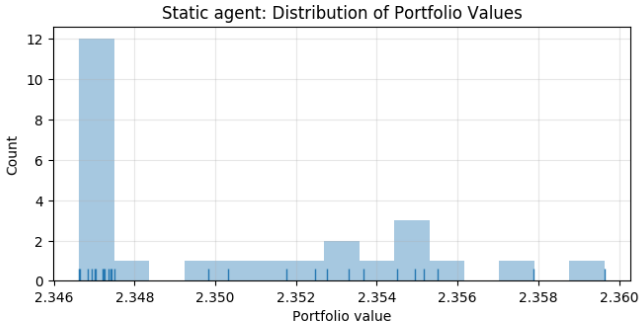
Dynamic agent	Average	Stdev
Ptf. value	2.0363	0.0026
Sharpe ratio	2.6533	0.0012
Sharpe ratio (ann)	7.1687	0.0033
MDD	0.4267	0.0018
Average of weights	0.0865	0.0004
Stdev of weights	0.0008	0.0
Cash weight (BTC)	0.0489	0.0043

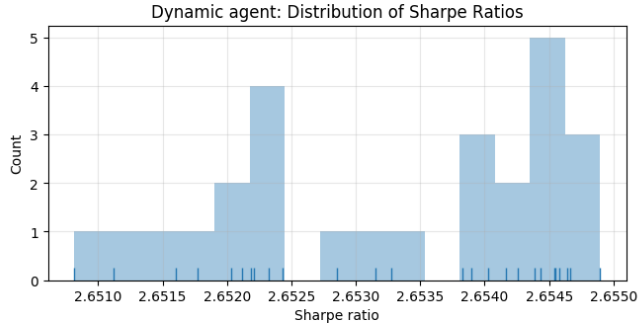
Static agent	Average	Stdev
Ptf. value	2.3505	0.0039
Sharpe ratio	2.6296	0.0013
Sharpe ratio (ann)	7.1047	0.0036
MDD	0.7653	0.0035
Average of weights	0.0865	0.0004
Stdev of weights	0.0008	0.0
Cash weight (BTC)	0.0489	0.0043

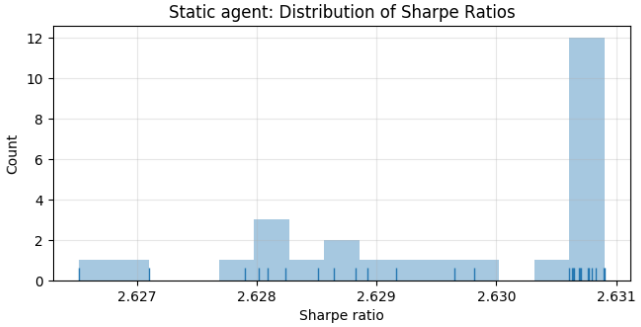
Equal weighted	Average
Ptf. value	2.3277
Sharpe ratio	2.6478
Sharpe ratio (ann)	7.1539
MDD	0.7399

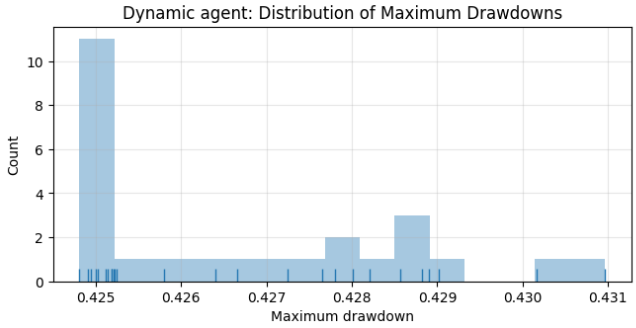
Parameter	Value
No. of simulations	26
No. of assets	11
Start date	2017-11-23
End date	2018-01-13
Trading period	15min

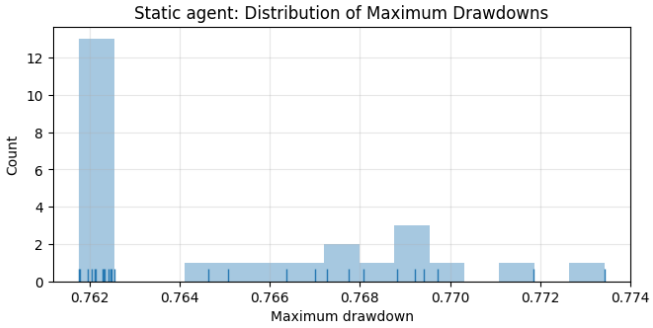


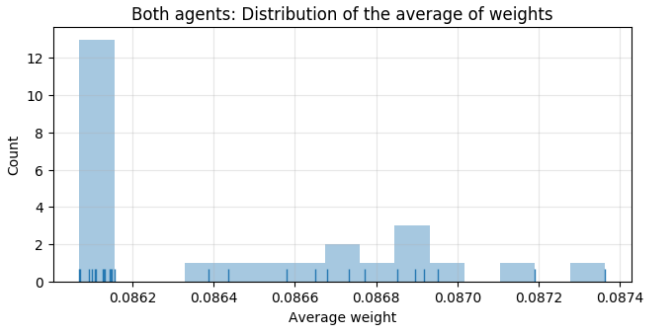


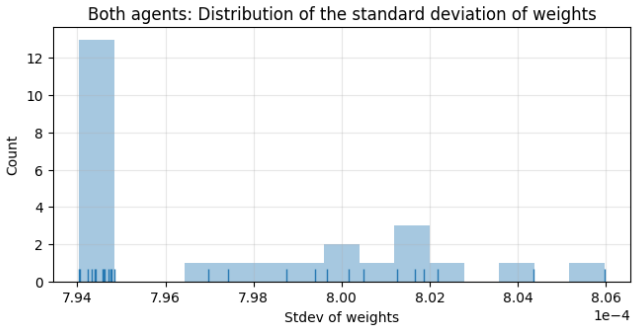












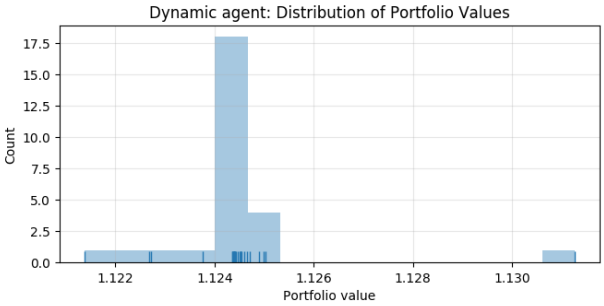
[Simulation summary] Rock bottom 2h

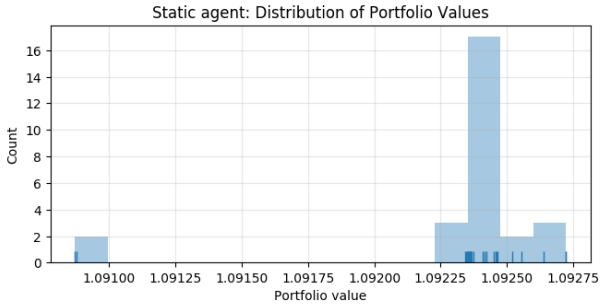
Dynamic agent	Average	Stdev
Ptf. value	1.1245	0.0015
Sharpe ratio	3.3298	0.016
Sharpe ratio (ann)	8.9966	0.0433
MDD	0.1197	0.0006
Average of weights	0.0873	0.0002
Stdev of weights	0.0016	0.0
Cash weight (BTC)	0.0392	0.0027

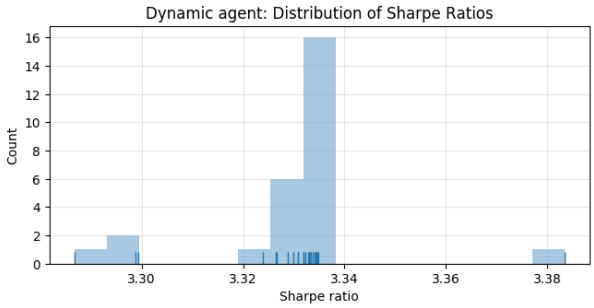
Static agent	Average	Stdev
Ptf. value	1.0923	0.0004
Sharpe ratio	2.3501	0.0099
Sharpe ratio (ann)	6.3496	0.0267
MDD	0.2006	0.0006
Average of weights	0.0873	0.0002
Stdev of weights	0.0016	0.0
Cash weight (BTC)	0.0392	0.0027

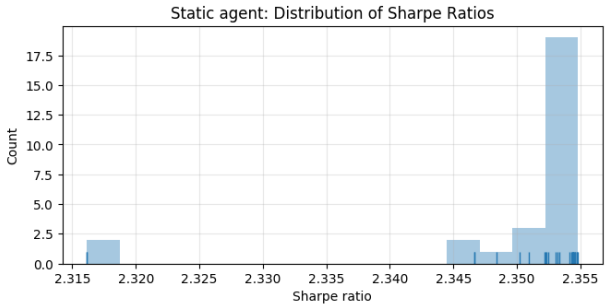
Equal weighted	Average
Ptf. value	1.0903
Sharpe ratio	2.4133
Sharpe ratio (ann)	6.7327
MDD	0.1923

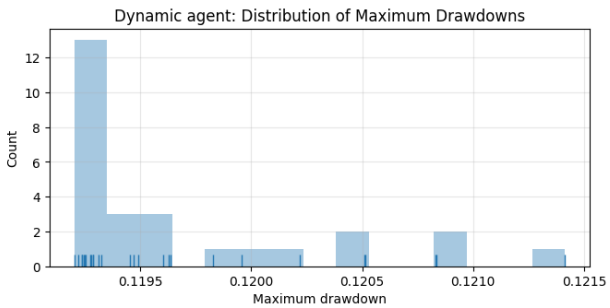
Parameter	Value
No. of simulations	27
No. of assets	11
Start date	2018-11-10
End date	2018-12-31
Trading period	2h

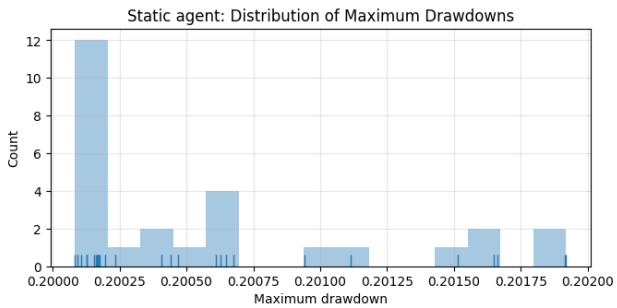


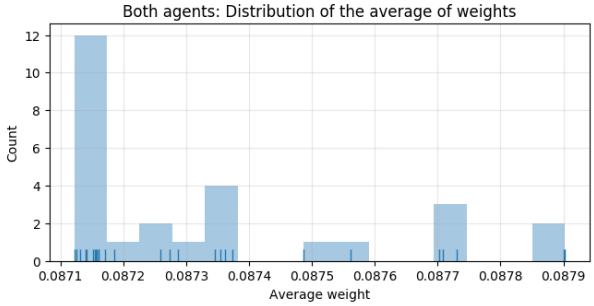


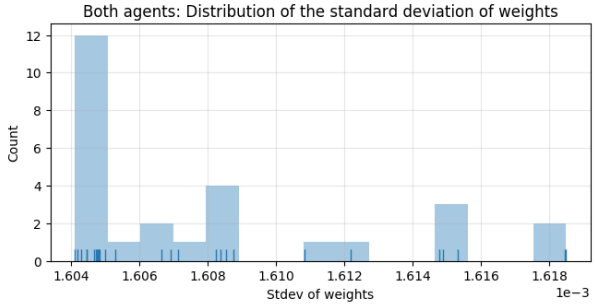












[Simulation stability] Recent 15min

Dynamic agent	Average	Stdev
Ptf. value	0.875	0.0119
Sharpe ratio	-2.7361	0.1352
Sharpe ratio (ann)	-7.3926	0.3653
MDD	0.2559	0.0087
Average of weights	0.0888	0.0023
Stdev of weights	0.0005	0.0
Cash weight (BTC)	0.0229	0.0252

Static agent	Average	Stdev
Ptf. value	0.8961	0.015
Sharpe ratio	-2.4056	0.2278
Sharpe ratio (ann)	-6.4995	0.6154
MDD	0.2638	0.0124
Average of weights	0.0888	0.0023
Stdev of weights	0.0005	0.0
Cash weight (BTC)	0.0229	0.0252

Equal weighted	Average
Ptf. value	0.9353
Sharpe ratio	-1.6941
Sharpe ratio (ann)	-4.5772
MDD	0.2351

Parameter	Value
No. of simulations	40
No. of assets	11
Start date	2019-03-06
End date	2019-04-26
Trading period	15min